Matheuristics

Hybridizing Metaheuristics and Mathematical Programming

Editor
Vittorio Maniezzo
Thomas Stützle
Stefan Voß

Volume 10

Annals of Information Systems

SPRINGER

Matheuristics

For other titles published in this series, go to
www.springer.com/series/7573

Annals of Information Systems

Volume 1:
Managing in the Information Economy: Current Research Issues
Uday Apte, Uday Karmarkar

Volume 2:
Decision Support for Global Enterprises
Uday Kulkarni, Daniel J. Power and Ramesh Sharda

Volume 3:
New Trends in Data Warehousing and Data Analysis
Stanislaw Kozielski, Robert Wremble

Volume 4:
Knowledge Management and Organizational Learning
William R. King

Volume 5:
Information Technology and Product Development
Satish Nambisan

Volume 6:
Web 2.0 & Semantic Web
Vladan Devedžic, Dragan Gaševic

Volume 7:
Web-Based Applications in Healthcare and Biomedicine
Athina Lazakidou

Volume 8:
Data Mining: Special Issue in Annals of Information Systems
Robert Stahlbock, Sven F. Crone and Stefan Lessmann

Volume 9:
Security Informatics
Christopher C. Yang, Michael Chiu-Lung Chau, Jau-Hwang Wang and Hsinchun Chen

Volume 10:
Matheuristics
Vittorio Maniezzo, Thomas Stützle and Stefan Voß

Matheuristics

Hybridizing Metaheuristics and Mathematical Programming

edited by

Vittorio Maniezzo
University of Bologna

Thomas Stützle
Université Libre de Bruxelles (ULB)

Stefan Voß
Universität Hamburg

Springer

Editors
Vittorio Maniezzo
Università di Bologna
Dept. Computer Science
Contrada Sacchi, 3
47023 Cesena
Italy
vittorio.maniezzo@unibo.it

Thomas Stützle
Université Libre de Bruxelles (ULB)
CoDE, IRIDIA, CP 194/6
Av. F. Roosevelt 50
1050 Brussels
Belgium
stuetzle@ulb.ac.be

Stefan Voß
Universität Hamburg
Dept. Wirtschaftwsissenschaften
Inst. Wirtschaftsinformatik
Von-Melle-Park 5
20146 Hamburg
Germany
stefan.voss@uni-hamburg.de

ISSN 1934-3221 e-ISSN 1934-3213
ISBN 978-1-4419-1305-0 e-ISBN 978-1-4419-1306-7
DOI 10.1007/978-1-4419-1306-7
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2009935392

Preface

The field of metaheuristics has traditionally been very receptive to proposals about how to structure algorithms in order to effectively solve optimization problems. Innovation of solution approaches has always been one of the traits of the field, and design paradigms have succeeded as inspiration for algorithm designers: inspiration from nature, improvements of local search, logics and probability, etc. The paradigm put forth in this book represents a "back to the roots" for computational optimization: use mathematics!

Albeit people working on metaheuristics have always been full citizens of the mathematical programming and operations research community and the main results have always been well-known, interactions (cross-fertilizations as it used to be fashionable to say) have always been limited. Core mathematical programming (MP) approaches have little to share with metaheuristics and mainstream metaheuristics include little or no mathematics.

This book shows how both metaheuristics and MP can leverage on one another. It shows how it is possible both to include MP techniques into metaheuristic frameworks and metaheuristic concepts inside MP systems. This follows a trend in hybridization, which appeared in several forms in the last years: metaheuristics are being hybridized with artificial intelligence, with constraint programming, with statistics, not to mention among themselves. However, the combination of metaheuristics and MP has a set-apart condition. Including MP techniques for a metaheuristic designer does not mean looking for contributions, which could possibly derive from another research area, it means looking inside one's own cultural baggage, it means using in a different way something one has already had experience with.

The contributions included in this collection comprise invited chapters that give an overview on specific topics in matheuristics and articles that were selected among the presentations at the *Matheuristics 2008* workshop. This was the second edition of a workshop series centered on the above ideas, having the stated objective of giving group identity to all researchers who share the interest in the synergies existing between the two related research lines metaheuristics and MP. The success of the first edition, which was upon

invitation only, suggested to open to submissions for the second. We scored an higher than 30% rejection rate at the conference, and acceptance barriers were high also for the present volume. We thus believe to have collected a set of good quality contributions, which will help in increasing the awareness of the possibilities offered by this new research direction.

The book includes 11 contributions, which span over a variety of topics.

Caserta and Voß open with an up-to-date overview of the field of metaheuristics, which helps to frame the matheuristics contributions in the more general context of metaheuristics advances.

Fischetti, Lodi and Salvagnin provide a survey on the use of mixed integer programming (MIP) solvers as subroutines for solving NP-hard subproblems, which arise while solving a more complex problem. Different success cases are reported, which follow this general idea.

Puchinger, Raidl and Pirkwieser review possibilities of how to use greedy heuristics, iterative improvement algorithms, and metaheuristics to improve the performance of MIP solvers. The possibilities for doing so are varied, and range from providing good quality starting solutions to using metaheuristics for cut separation or column generation.

Dumitrescu and Stützle review approaches that combine local search and metaheuristics with MP techniques. In particular, they focus on algorithms where the metaheuristic is the master solver and MP techniques are used to solve subproblems arising in the search process.

Boschetti, Maniezzo and Roffilli show how it is possible to use decomposition techniques, which were originally conceived as tools for exact optimization, as metaheuristic frameworks. The general structure of each of the best known decomposition approaches (Lagrangean, Benders and Dantzig-Wolfe) can, in fact, be considered as a general structure for a corresponding metaheuristic.

Gutjahr proposes a more theoretical contribution. His chapter gives an overview on techniques for proving convergence of metaheuristics to optimal or sufficiently good solutions. Two particularly interesting topics that are covered are convergence of metaheuristic algorithms for stochastic combinatorial optimization, and estimation of expected runtime, i.e., of the time needed by a metaheuristic to hit the first time a solution of a required quality.

The remaining papers address specific problems by means of matheuristic algorithms, rather than presenting general overviews as the ones above.

Dolgui, Eremeev and Guschinskaya address the problem of balancing transfer lines with multi-spindle machines. They present two algorithms that integrate traditional metaheuristics and a MIP solver. Specifically, they design a GRASP and a genetic algorithm, which both use a MIP solver as a subroutine for solving subproblems arising in the search process of the metaheuristics.

Gruber and Raidl work on exact algorithms for the bounded diameter minimum spanning tree problem. They use simple heuristics and a tabu search algorithm for solving the separation problem in their branch-and-cut algo-

rithm. Moreover, they include in their approach a variable neighborhood descent for finding good primal solutions.

Liberti, Nannicini and Mladenović present a work in mixed integer nonlinear programming, having the objective of identifying good quality, or at least feasible solutions for difficult instances. They propose a method, called RECIPE (for Relaxed-Exact Continuous-Integer Problem Exploration), combining variable neighborhood search, local branching, sequential quadratic programming and branch-and-bound.

Mitrović-Minić and Punnen present their method consisting in a local search where large neighborhoods are explored by means of ancillary MIP subproblems. They present results on the basic generalized assignment problem (GAP) and on the multi-resource GAP showing the potential of the approach.

Finally, Ulrich-Ngueveu, Prins, and Wolfler-Calvo study the m-peripatetic vehicle routing problem, which is a special vehicle routing problem, asking that each arc is used in the solution at most once for each set of m periods considered in the plan. The approach uses a perfect b-matching to define the candidate sets used in a granular tabu search algorithm.

We conclude expressing our gratitude to all authors who submitted their works to Matheuristics 2008 and to this post-conference collection, to the members of the international program committee and to all external reviewers. We are confident that this book, as a result of their joint effort, will provide a basis to support the increasing interest on the covered topics. We hope and believe that the result constitutes a significant achievement in the direction of establishing matheuristics as a credible tool for obtaining fast and reliable solutions to real-world problems.

Bologna, Brussels, Hamburg,
April 2009

Vittorio Maniezzo
Thomas Stützle
Stefan Voß

Contents

List of Contributors

Marco Boschetti
Department of Mathematics, Università di Bologna, Bologna, Italy
e-mail: marco.boschetti@unibo.it

Marco Caserta
Institute of Information Systems (Wirtschaftsinformatik), University of Hamburg, Hamburg, Germany
e-mail: marco.caserta@uni-hamburg.de

Alexandre Dolgui
Department Scientific Methods for Industrial Management, Ecole Nationale Supérieure des Mines de Saint Etienne, Saint Etienne, France
e-mail: dolgui@emse.fr

Irina Dumitrescu
School of Mathematics, University of New South Wales, Sydney, Australia
e-mail: irina.dumitrescu@unsw.edu.au

Anton Eremeev
Omsk Branch of Sobolev Institute of Mathematics, Omsk, Russia
e-mail: eremeev@ofim.oscsbras.ru

Matteo Fischetti
DEI, Università di Padova, Padua, Italy
e-mail: matteo.fischetti@unipd.it

Martin Gruber
Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria
e-mail: gruber@ads.tuwien.ac.at

Olga Guschinskaya
Department Scientific Methods for Industrial Management, Ecole Nationale Supérieure des Mines de Saint Etienne, Saint Etienne, France
e-mail: guschinskaya@emse.fr

Walter J. Gutjahr
Department of Statistics and Decision Support Systems, University of Vienna, Vienna, Austria
e-mail: walter.gutjahr@univie.ac.at

Andrea Lodi
DEIS, Università di Bologna, Bologna, Italy
e-mail: andrea.lodi@unibo.it

Leo Liberti
LIX, École Polytechnique, Palaiseau, France
e-mail: liberti@lix.polytechnique.fr

Vittorio Maniezzo
Department of Computer Science, Università di Bologna, Cesena, Italy
e-mail: vittorio.maniezzo@unibo.it

Snežana Mitrović-Minić
Department of Mathematics, Simon Fraser University, Surrey, Canada
e-mail: snezanam@sfu.ca

Nenad Mladenović
Department of Mathematical Sciences, Brunel University, London, UK and Institute of Mathematics, Academy of Sciences, Belgrade, Serbia
e-mail: nenad.mladenovic@brunel.ac.uk,nenad@turing.mi.sanu.ac.yu

Giacomo Nannicini
LIX, École Polytechnique, Palaiseau, France
e-mail: giacomon@lix.polytechnique.fr

Sandra Ulrich Ngueveu
Institut Charles Delaunay—LOSI, Université de Technologie de Troyes (UTT), Troyes, France
e-mail: ngueveus@utt.fr

Sandro Pirkwieser
Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria
e-mail: pirkwieser@ads.tuwien.ac.at

Christian Prins
Institut Charles Delaunay—LOSI, Université de Technologie de Troyes (UTT), Troyes, France
e-mail: christian.prins@utt.fr

Jakob Puchinger
arsenal research, Vienna, Austria
e-mail: jakob.puchinger@arsenal.ac.at

Abraham P. Punnen
Department of Mathematics, Simon Fraser University, Surrey, Canada
e-mail: apunnen@sfu.ca

Günther R. Raidl
Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria
e-mail: raidl@ads.tuwien.ac.at

Matteo Roffilli
Department of Computer Science, University of Bologna, Cesena, Italy
e-mail: roffilli@csr.unibo.it

Domenico Salvagnin
DMPA, Università di Padova, Padua, Italy
e-mail: dominiqs@gmail.com

Thomas Stützle
IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium
e-mail: stuetzle@ulb.ac.be

Stefan Voß
Institute of Information Systems (Wirtschaftsinformatik), University of Hamburg, Hamburg, Germany
e-mail: stefan.voss@uni-hamburg.de

Roberto Wolfler Calvo
Institut Charles Delaunay—LOSI, Université de Technologie de Troyes (UTT), Troyes, France
e-mail: roberto.wolfler_calvo@utt.fr

Chapter 1
Metaheuristics: Intelligent Problem Solving

Marco Caserta and Stefan Voß

Abstract Metaheuristics support managers in decision making with robust tools providing high quality solutions to important problems in business, engineering, economics and science in reasonable time horizons. While finding exact solutions in these applications still poses a real challenge despite the impact of recent advances in computer technology and the great interactions between computer science, management science/operations research and mathematics, (meta-) heuristics still seem to be the methods of choice in many (not to say most) applications. In this chapter we give some insight into the state of the art of metaheuristics. It focuses on the significant progress regarding the methods themselves as well as the advances regarding their interplay and hybridization with exact methods.

1.1 Introduction

The use of heuristics and metaheuristics to solve real world problems is widely accepted within the operations research community. It is well-known that the great majority of complex real world decision problems, when modeled as optimization problems, belong to the class of $\mathcal{NP}$-hard problems. This implies that exact approaches are doomed to fail when dealing with large scale instances, whether they arise from business, engineering, economics or science. Today, decision making processes are increasingly complex and more encompassing, in the sense that more decision variables are used to model complex situations and more input data and parameters are available to capture the complexity of the problems themselves.

Marco Caserta · Stefan Voß
Institute of Information Systems (Wirtschaftsinformatik), University of Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany
e-mail: {marco.caserta,stefan.voss}@uni-hamburg.de

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_1,

The inherent complexity of real world optimization problems, though, should be interpreted in the light of what complexity analysis really means: on the one hand, the fact that a problem belongs to the class of $\mathcal{NP}$-hard problems implies that there is no knowledge of an algorithm capable of solving the problem itself to optimality in polynomial time with respect to its input size (and many believe that there will never be); on the other hand, it is worth remembering that complexity analysis provides a worst case scenario, i.e., it indicates that, in the worst case, with the growth of the input size, the algorithm will require more and more time/steps to provide a definite answer. However, it is also worth noting that practitioners have observed that it is possible to design *ad-hoc* algorithms which, while not guaranteeing optimal or near-optimal solutions for the whole set of possible instances of a given problem (and, in many cases, not even a simple, single answer can be guaranteed at all), they do provide near-optimal solutions "most" of the times. This is exactly the goal of a metaheuristic designer (analyst), namely, to design an algorithm for which, even though no guarantee about the worst case scenario can be offered, a certain degree of confidence about the performance of the algorithm "most of the times" can still be asserted.

Consequently, the real challenge of the metaheuristic expert is not only to objectively measure the algorithm in terms of solution quality and computational time over problems for which no optimal solution is known (which, in itself, can be a challenging task due to the lack of any benchmark) but also to use sound quantitative methods and techniques to assert the robustness of the algorithm over a wide spectrum of instance types, hence enhancing the usability of the algorithm itself by industry decision makers, e.g., as "optimization modules" within decision support systems.

A first trade-off emerges here: on the one hand, it is highly desirable to design "general purpose" metaheuristics, which do not require problem specific knowledge and can readily be applied to a wide spectrum of problem classes. This has been the line of research of the last decades, during which a number of general purpose metaheuristic paradigms have been proposed, e.g., simulated annealing, genetic algorithms, tabu search, ant colony, etc. The main argument in favor of such paradigms is exactly their general applicability upon a large set of problems, without requiring major re-design or any in-depth knowledge of the problem to be tackled. Consequently, general paradigms seem especially suited for practitioners, who are interested in getting a solution to the problem without investing a huge amount of time in understanding the mathematical properties of the model and in implementing tailor-made algorithms. However, most successful metaheuristics applied to specific problems are also tailored to the problem, or fine-tuned. On the other hand, it has been observed that general purpose metaheuristics are outperformed by hybrid algorithms, which usually are algorithms that combine mathematical programming techniques with metaheuristic-based ideas. Algorithms of this class are tailor-made and specifically designed to exploit the mathematical properties of the problem at hand. While such approaches are

able to provide enhanced performance, an obvious consequence is a reduction in the usability of the algorithm itself. In general, a tailor-made algorithm can be used only for a specific class of problems and, often, the underlying ideas cannot easily be extended towards operating on a different class of problems.

The discussion on metaheuristic based algorithms is further complicated by what has been observed in recent years: these algorithms in general seem to provide varying performance depending upon the sensitivity (skills, expertise, ingenuity, etc.) of the designer in algorithmic fine tuning. In order to maximize algorithmic performance, an instance specific fine tuning might be required. The variability in the results of a metaheuristic presents at least two major drawbacks: (i) On one hand, the issue of *reproducibility* of results arises: It is common knowledge that a proposed algorithm implemented by two different researchers can lead to altogether different results, depending upon factors such as implementation skills, special usage of data structures, and ability in parameter settings, among others. For this reason, it is hard to compare and thoroughly assess a metaheuristic and its performance. (ii) On the other hand, a problem of *performance maximization* is envisioned: One of the commonalities of virtually all proposed metaheuristic paradigms is that they are characterized by a considerable number of parameters, whose value(s) strongly affect the overall performance of the algorithm itself. It is common knowledge that the fine tuning of algorithmic parameters is not only problem-specific, but even instance-specific, i.e., even for a given problem, parameter values should be fine-tuned and adjusted according to instance specific information (e.g., instance size, distribution of its values, etc.).

The purpose of this paper is to provide a survey of the general field of metaheuristics. While we cannot be fully comprehensive in a single paper, in line with the above remarks, some of the issues addressed in this paper are:

- In light of the well-known *no-free-lunch-theorem* [128], which basically states that, on average, no algorithm outperforms all the others, one might wonder what strategy to pursue, i.e., whether the goal of a researcher in the field should be to develop a better general framework able to effectively solve a wider spectrum of problems or, conversely, to tackle each individual problem separately, by designing tailor-made algorithms that fully exploit the mathematical structure and properties of each problem. A recent line of research in the metaheuristic field is concerned with the design of *hybrid* algorithms, where the term hybrid can indicate either the combination of different metaheuristics or the intertwined usage of metaheuristic features with mathematical programming techniques. Consequently, a trade-off between *re-usability* and *performance* of an algorithm arises.
- As highlighted before, no heuristic can guarantee high quality solutions over all possible instances of a given problem class. However, it is at least desirable to present a *robust* behavior over a spectrum of instances belonging to the same problem class. One key factor that seems to have a strong impact on the algorithmic performance is the fine tuning of the algorithm itself. Since the behavior of a metaheuristic is affected by its parameters,

one might wonder how to select a good set of parameter values. An important topic in metaheuristic design is, therefore, the identification and development of techniques for the *fine tuning of algorithmic parameters.*

- Due to the stochastic behavior of some metaheuristics and the lack of commonly accepted and adopted techniques for the evaluation of algorithmic performance, given two algorithms designed to tackle the same class of problems, it is not always possible to "rank" such algorithms in terms of their performance. One direct consequence is that there is still no clear understanding about which features are really successful and under which circumstances. In other words, unless clear standards about metaheuristics are defined, it will be really hard to have a full grasp of, first, which algorithms are better than others and, second, what is the real contribution of each feature of the algorithm upon the overall performance.

The structure of this paper is as follows: We first present general concepts about heuristics and metaheuristics, seen from an operations research perspective. Next, we illustrate some findings from recent research about hybridization, seen as a combination of exact approaches and mathematical programming techniques with metaheuristic frameworks. We also include here some recent contributions about metaheuristic design, i.e., a collection of ideas and thoughts about what should influence which features of a metaheuristic paradigm to be included in the algorithm (e.g., fitness landscape evaluation). Next, we focus on the important issue of metaheuristics fine tuning and calibrations, by introducing some quantitative methods drawn from statistics that have been employed with this goal in mind. We also present some thoughts on the ongoing debate on metaheuristic assessment, *i.e.*, how an objective evaluation of the performance of a metaheuristic can be carried out in order to increase objectivity of the evaluation and reproducibility of the results. Next, we mention some optimization software libraries especially focused on the implementation of general heuristic and metaheuristic frameworks. The development of software libraries for metaheuristics is perceived as a key factor in the emerging of standards in the field. Finally, the last section presents some concluding remarks.

Earlier survey papers on metaheuristics include [19, 121, 122].[1] The general concepts have not become obsolete, and many changes are mainly based upon an update to most recent references. A handbook on metaheuristics is available describing a great variety of concepts by various authors in a comprehensive manner [59].

[1] Here we occasionally rely on [121] and [122] without explicitly quoting at appropriate places for not "disturbing" the readability.

1.2 Basic Concepts and Discussion

The basic concept of heuristic search as an aid to problem solving was first introduced by [93]. A *heuristic* is a technique (consisting of a rule or a set of rules) which seeks (and hopefully finds) *good* solutions at a reasonable computational cost. A heuristic is *approximate* in the sense that it provides (hopefully) a good solution for relatively little effort, but it does not guarantee optimality.

Heuristics provide simple means of indicating which among several alternatives seems to be best. That is, "heuristics are criteria, methods, or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal. They represent compromises between two requirements: the need to make such criteria simple and, at the same time, the desire to see them discriminate correctly between good and bad choices. A heuristic may be a *rule of thumb* that is used to guide one's action." [91]

Greedy heuristics are simple iterative approaches available for any kind of (e.g., combinatorial) optimization problem. A good characterization is their *myopic* behavior. A greedy heuristic starts with a given feasible or infeasible solution. In each iteration there is a number of alternative choices (*moves*) that can be made to transform the solution. From these alternatives which consist in fixing (or changing) one or more variables, a *greedy choice* is made, i.e., the best alternative according to a given measure is chosen until no such transformations are possible any longer.

Usually, a greedy *construction heuristic* starts with an incomplete solution and completes it in a stepwise fashion. Savings and dual algorithms follow the same iterative scheme: Dual heuristics change an infeasible low cost solution until reaching feasibility, savings algorithms start with a high cost solution and realize the highest savings as long as possible. Moreover, in all three cases, once an element is chosen this decision is (usually) not reversed throughout the algorithm, it is kept.

As each alternative has to be measured, in general we may define some sort of *heuristic measure* (providing, e.g., some priority values or some ranking information) which is iteratively followed until a complete solution is build. Usually this heuristic measure is applied in a greedy fashion.

For heuristics we usually have the distinction between finding initial feasible solutions and improving them. In that sense we first discuss local search before characterizing metaheuristics.

1.2.1 Local Search

The basic principle of local search is to successively alter solutions *locally*. Related transformations are defined by neighborhoods which for a given solution

include all solutions that can be reached by one move. That is, neighborhood search usually is assumed to proceed by moving iteratively from one solution to another one by performing some sort of operation. More formally, each solution of a problem has an associated set of neighbors called its *neighborhood*, i.e., solutions that can be obtained by a single operation called transformation or move. Most common ideas for transformations are, e.g., to add or drop some problem specific individual components. Other options are to exchange two components simultaneously, or to swap them. Furthermore, components may be shifted from a certain position into other positions. All components involved within a specific move are called its elements or attributes.

Moves must be evaluated by some *heuristic measure* to guide the search. Often one uses the implied change of the objective function value, which may provide reasonable information about the (local) advantage of moves. Following a greedy strategy, *steepest descent* (SD) corresponds to selecting and performing in each iteration the best move until the search stops at a local optimum. Obviously, savings algorithms correspond to SD.

As the solution quality of local optima may be unsatisfactory, we need mechanisms that guide the search to overcome local optimality. For example, a metaheuristic strategy called iterated local search is used to iterate/restart the local search process after a local optimum has been obtained, which requires some perturbation scheme to generate a new initial solution (e.g., performing some random moves). Of course, more structured ways to overcome local optimality may be advantageous.

A general survey on local search can be found in [1] and the references from [2]. A simple template is provided by [116].

Despite the first articles on this topic already being in the 1970s (cf. Lin and Kernighan [82]), a variable way of handling neighborhoods is still a topic within local search. Consider an arbitrary neighborhood structure N, which defines for any solution s a set of neighbor solutions $N_1(s)$ as a neighborhood of depth $d = 1$. In a straightforward way, a neighborhood $N_{d+1}(s)$ of depth $d + 1$ is defined as the set $N_d(s) \cup \{s' | \exists s'' \in N_d(s) : s' \in N_1(s'')\}$. In general, a large d might be unreasonable, as the neighborhood size may grow exponentially. However, depths of two or three may be appropriate. Furthermore, temporarily increasing the neighborhood depth has been found to be a reasonable mechanism to overcome *basins of attraction*, e.g., when a large number of neighbors with equal quality exist [12, 13].

Large scale neighborhoods and large scale neighborhood search have become an important topic (see, e.g., [5] for a survey), especially when efficient ways are at hand for exploring them. Related research can also be found under various names; see, e.g., [92] for the idea of ejection chains.

1.2.2 Metaheuristics

The formal definition of metaheuristics is based on a variety of definitions from different authors derived from [52]. Basically, a metaheuristic is a top-level strategy that guides an underlying heuristic solving a given problem. In that sense we distinguish between a *guiding process* and an *application process*. The guiding process decides upon possible (local) moves and forwards its decision to the application process which then executes the chosen move. In addition, it provides information for the guiding process (depending on the requirements of the respective metaheuristic) like the recomputed set of possible moves.

According to [58] "metaheuristics in their modern forms are based on a variety of interpretations of what constitutes *intelligent search*," where the term *intelligent search* has been made prominent by Pearl [91] (regarding heuristics in an artificial intelligence context; see also [118] regarding an operations research context). In that sense we may also consider the following definition: "A metaheuristic is an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search spaces using learning strategies to structure information in order to find efficiently near-optimal solutions." [88].

To summarize, the following definition seems to be most appropriate: *"A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method. The family of metaheuristics includes, but is not limited to, adaptive memory procedures, tabu search, ant systems, greedy randomized adaptive search, variable neighborhood search, evolutionary methods, genetic algorithms, scatter search, neural networks, simulated annealing, and their hybrids."* [124], p. ix.

1.2.2.1 Simulated Annealing

Simulated annealing (SA) extends basic local search by allowing moves to inferior solutions [80, 35]. A basic SA algorithm may be described as follows: Iteratively, a candidate move is randomly selected; it is accepted if it leads to a solution with an improved objective function value compared to the current solution. Otherwise, the move is accepted with a probability depending on the deterioration Δ of the objective function value. The acceptance probability is computed as $e^{-\Delta/T}$, using a temperature T as control parameter. Usually, T is reduced over time for diversification at an earlier stage of the search and to intensify later.

Various authors describe a robust implementation of this general SA approach; see, e.g., [79], [77]. An interesting variant of SA is to strategically reheat the process, i.e., to perform a non-monotonic acceptance function.

Threshold accepting [37] is a modification (or simplification) of SA accepting every move that leads to a new solution that is 'not much worse' than the older one (i.e., it deteriorates not more than a certain threshold, which reduces with a temperature).

1.2.2.2 Tabu Search

The basic paradigm of *tabu search* (TS) is to use information about the search history to guide local search approaches to overcome local optimality (see [58] for a survey on TS). In general, this is done by a dynamic transformation of the local neighborhood. Based on some sort of memory, certain moves may be forbidden, i.e., they are set tabu. As for SA, the search may lead to performing deteriorating moves when no improving moves exist or all improving moves of the current neighborhood are set tabu. At each iteration, a best admissible neighbor may be selected. A neighbor, respectively a corresponding move, is called admissible, if it is not tabu or if an aspiration criterion is fulfilled. An aspiration criterion is a rule to eventually override a possibly unreasonable tabu status of a move. For example, a move that leads to a neighbor with a better objective function value than encountered so far should be considered as admissible.

The most commonly used TS method is based on a *recency-based* memory that stores moves, or attributes characterizing respective moves, of the recent past (*static TS*). The basic idea of such approaches is to prohibit an appropriately defined inversion of performed moves for a given period by storing attributes of the solution in a tabu list and then preventing moves that require the use of attributes in such a list.

Strict TS embodies the idea of preventing cycling to formerly traversed solutions. The goal is to provide necessity and sufficiency with respect to the idea of not revisiting any solution. Accordingly, a move is classified as tabu iff it leads to a neighbor that has already been visited during the previous search. There are two primary mechanisms to accomplish the tabu criterion: First, we may exploit logical interdependencies between the sequence of moves performed throughout the search process, as realized by, e.g., the reverse elimination method (cf., e.g., [53, 120]). Second, we may store information about all solutions visited so far. This may be carried out either exactly or, for reasons of efficiency, approximately (e.g., by using hash codes).

Reactive TS aims at the automatic adaptation of the tabu list length of static TS [15] by increasing the tabu list length when the tabu memory indicates that the search is revisiting formerly traversed solutions. A possible specification can be described as follows: Starting with a tabu list length l of 1 it is increased every time a solution has been repeated. If there has been

no repetition for some iterations, we decrease it appropriately. To accomplish the detection of a repetition of a solution, one may apply a trajectory based memory using hash codes as for strict TS.

There is a large number of additional ingredients that may make TS work well. Examples include restricting the number of neighbor solutions to be evaluated (using candidate list strategies, e.g., [97]), logical tests as well as diversification mechanisms.

1.2.2.3 Evolutionary Algorithms

Evolutionary algorithms comprise a great variety of different concepts and paradigms including genetic algorithms (see, e.g., [73, 60]), evolutionary strategies (see, e.g., [72, 106]), evolutionary programs [48], scatter search (see, e.g., [51, 54]), and memetic algorithms [87]. For surveys and references on evolutionary algorithms see also [49, 9, 85, 99].

Genetic algorithms are a class of adaptive search procedures based on principles derived from the dynamics of natural population genetics. One of the most crucial ideas for a successful implementation of a genetic algorithm (GA) is the representation of an underlying problem by a suitable scheme. A GA starts, e.g., with a randomly created initial population of artificial creatures (strings), a set of solutions. These strings in whole and in part are the base set for all subsequent populations. Information is exchanged between the strings in order to find new solutions of the underlying problem. The mechanisms of a simple GA essentially consist of copying strings and exchanging partial strings. A simple GA uses three operators which are named according to the corresponding biological mechanisms: reproduction, crossover, and mutation. Performing an operator may depend on a *fitness function* or its value (fitness), respectively. As some sort of heuristic measure, this function defines a means of measurement for the profit or the quality of the coded solution for the underlying problem and often depends on the objective function of the given problem.

GAs are closely related to *evolutionary strategies*. Whereas the mutation operator in a GA was argued to serve to protect the search from premature loss of information [60], evolutionary strategies may incorporate some sort of local search procedure (such as SD) with self adapting parameters involved in the procedure. On a simplified scale many algorithms may be classified as evolutionary once they are reduced to the following frame (see [71]). First, an initial population of individuals is generated. Second, as long as a termination criterion does not hold, perform some sort of co-operation and self-adaptation. Self-adaptation refers to the fact that individuals (solutions) evolve independently while co-operation refers to an information exchange among individuals.

Scatter search ideas established a link between early ideas from various sources—evolutionary strategies, TS and GAs. As an evolutionary approach,

scatter search originated from strategies for creating composite decision rules and surrogate constraints (see [51]). Scatter search is designed to operate on a set of points, called reference points, that constitute good solutions obtained from previous solution efforts. The approach systematically generates linear combinations of the reference points to create new points, each of which is mapped into an associated point that yields integer values for discrete variables. Scatter search contrasts with other evolutionary procedures, such as GAs, by providing unifying principles for joining solutions based on generalized path constructions in Euclidean space and by utilizing strategic designs where other approaches resort to randomization. For a very comprehensive treatment of scatter search see [81].

1.2.2.4 Cross Entropy Method

Initially proposed by [104] for the estimation of rare events, the *cross entropy* method (CE) was extended to solve combinatorial optimization problems [28]. The key ingredient in the CE is the identification of a parametric probability distribution function to be used to generate feasible solutions. Given an initial probability distribution function ϕ^0, a converging sequence of ϕ^t is generated in such a way that each subsequent probability distribution function better captures prominent features found in high quality solutions.

At any given iteration t, ϕ^t is used to generate a population of a given cardinality. Each solution is then evaluated according to a specified merit function (or heuristic measure), e.g., the objective function value associated to each random variate, and the stochastic parameters are then updated in such a way that, in the next generation of the population, high quality solutions will have higher probabilities of being generated under the new model. The problem of updating the stochastic parameters can be solved by applying the *maximum likelihood estimator* method upon a set of "elite solutions" of the current population. In other words, given the top $\rho\%$ of the current population, the CE aims at identifying the value of the parameters of the probability distribution function that better "explains" these elite solutions. This corresponds to adjusting the model to better describe the portion of the feasible space in which good solutions have been found. The two-phase process of generation and update is repeated until convergence in probability is reached.

1.2.2.5 Ant Colony Optimization

The *ant colony optimization* (ACO) metaheuristic [32, 33] is a stochastic method based upon the definition of a construction graph and the use of a set of stochastic procedures called artificial ants. A number of frameworks for the update of stochastic parameters have been proposed. The *ant system*

is a dynamic optimization process reflecting the natural interaction between ants searching for food (see, e.g., [32, 33]). The ants' ways are influenced by two different kinds of search criteria. The first one is the local visibility of food, i.e., the attractiveness of food in each ant's neighborhood. Additionally, each ant's way through its food space is affected by the other ants' trails as indicators for possibly good directions. The intensity of trails itself is time-dependent: With time passing, parts of the trails are diminishing, meanwhile the intensity may increase by new and fresh trails. With the quantities of these trails changing dynamically, an autocatalytic optimization process is started forcing the ants' search into most promising regions. This process of interactive learning can easily be modeled for most kinds of optimization problems by using simultaneously and interactively processed search trajectories.

A comprehensive treatment of the ant system paradigm can be found in [33]. To achieve enhanced performance of the ant system it is useful to hybridize it at least with a local search component.

As pointed out by [133], there are a number of commonalities between the CE and the ACO method, especially with respect to the parameter updating rule mechanisms.

1.2.2.6 Corridor Method

The *corridor method* (CM) has been presented by [109] as a hybrid metaheuristic, linking together mathematical programming techniques with heuristic schemes. The basic idea of the CM relies on the use of an exact method over restricted portions of the solution space of a given problem. Given an optimization problem P, the basic ingredients of the method are a very large feasible space $\mathcal{X}$, and an exact method M that could easily solve problem P if the feasible space were not large. Since, in order to be of interest, problem P generally belongs to the class of $\mathcal{NP}$-hard problems, the direct application of method M to solve P becomes unpractical when dealing with real world instances, i.e., when $\mathcal{X}$ is large.

The basic concept of a *corridor* is introduced to delimit a portion of the solution space around the incumbent solution. The optimization method will then be applied within the neighborhood defined by the corridor with the aim of finding an improved solution. Consequently, the CM defines method-based neighborhoods, in which a neighborhood is built taking into account the method M used to explore it. Given a current feasible solution $\mathbf{x} \in \mathcal{X}$, the CM builds a neighborhood of $\mathbf{x}$, say $\mathcal{N}(\mathbf{x})$, which can effectively be explored by employing method M. Ideally, $\mathcal{N}(\mathbf{x})$ should be exponentially large and built in such a way that it could be explored in (pseudo) polynomial time using method M.

1.2.2.7 Pilot Method

Building on a simple greedy algorithm such as, e.g., a construction heuristic the *pilot method* [38, 39] is a metaheuristic not necessarily based on a local search in combination with an improvement procedure. It primarily *looks ahead* for each possible local choice (by computing a so-called "pilot" solution), memorizing the best result, and performing the respective move. (Very similar ideas have been investigated under the acronym *rollout method* [17].) One may apply this strategy by successively performing a greedy heuristic for all possible local steps (i.e., starting with all incomplete solutions resulting from adding some not yet included element at some position to the current incomplete solution). The look ahead mechanism of the pilot method is related to increased neighborhood depths as the pilot method exploits the evaluation of neighbors at larger depths to guide the neighbor selection at depth one.

In most applications, it is reasonable to restrict the pilot process to some *evaluation depth*. That is, the method is performed up to an incomplete solution (e.g., partial assignment) based on this evaluation depth and then completed by continuing with a conventional heuristic. For a recent study applying the pilot method to several combinatorial optimization problems obtaining very good results see [123]. Additional applications can be found, e.g., in [84, 22].

1.2.2.8 Other Methods

*A*daptive memory programming (AMP) coins a general approach (or even thinking) within heuristic search focusing on exploiting a collection of memory components [55, 114]. An AMP process iteratively constructs (new) solutions based on the exploitation of some memory, especially when combined with learning mechanisms supporting the collection and use of the memory. Based on the idea of initializing the memory and then iteratively generating new solutions (utilizing the given memory) while updating the memory based on the search, we may subsume various of the above described metaheuristics as AMP approaches. This also includes exploiting provisional solutions that are improved by a local search approach.

The performance as well as the efficiency of a heuristic scheme strongly depends on its ability to use AMP techniques providing flexible and variable strategies for types of problems (or special instances of a given problem type) where standard methods fail. Such AMP techniques could be, e.g., dynamic handling of operational restrictions, dynamic move selection formulas, and flexible function evaluations.

Consider, as an example, adaptive memory within TS concepts. Realizing AMP principles depends on which specific TS application is used. For example, the reverse elimination method observes logical interdependencies be-

tween moves and infers corresponding tabu restrictions, and therefore makes fuller use of AMP than simple static approaches do.

To discuss the use of AMP in intelligent agent systems, one may use the simple model of ant systems as an illustrative starting point. As ant systems are based on combining constructive criteria with information derived from the pheromone trails, this follows the AMP requirement for using flexible (dynamic) move selection rules (formulas). However, the basic ant system exhibits some structural inefficiencies when viewed from the perspective of general intelligent agent systems, as no distinction is made between successful and less successful agents, no time-dependent distinction is made, there is no explicit handling of restrictions providing protection against cycling and duplication. Furthermore, there are possible conflicts between the information held in the adaptive memory (*diverging trails*).

A natural way to solve large optimization problems is to decompose them into independent sub-problems that are solved with an appropriate procedure. However, such approaches may lead to solutions of moderate quality since the sub-problems might have been created in a somewhat arbitrary fashion. Of course, it is not easy to find an appropriate way to decompose a problem *a priori*. The basic idea of POPMUSIC is to locally optimize subparts of a solution, *a posteriori*, once a solution to the problem is available. These local optimizations are repeated until a local optimum is found. Therefore, POPMUSIC may be viewed as a local search working with a special, large neighborhood. While POPMUSIC has been acronymed by [112] other metaheuristics may be incorporated into the same framework, too (e.g. [107]). Similarly, in the *variable neighborhood search* (VNS) [68] the neighborhood is altered during the search in such a way that different, e.g. increasingly distant, neighborhoods of a given solution are explored. Such method can be enhanced via *decomposition*, as in the *variable neighborhood decomposition search* (VNDS) (see, e.g., [69]).

For large optimization problems, it is often possible to see the solutions as composed of parts (or chunks [129], cf. the term vocabulary building). Considering as an example the vehicle routing problem, a part may be a tour (or even a customer). Suppose that a solution can be represented as a set of parts. Moreover, some parts are more in relation with some other parts so that a corresponding heuristic measure can be defined between two parts. The central idea of POPMUSIC is to select a so-called *seed part* and a set P of parts that are mostly related with the seed part to form a sub-problem.

Then it is possible to state a local search optimization frame that consists of trying to improve all sub-problems that can be defined, until the solution does not contain a sub-problem that can be improved. In the POPMUSIC frame of [112], the set of parts P corresponds precisely to seed parts that have been used to define sub-problems that have been unsuccessfully optimized. Once P contains all the parts of the complete solution, then all sub-problems have been examined without success and the process stops.

Basically, the technique is a gradient method that starts from a given initial solution and stops in a local optimum relative to a large neighborhood structure. To summarize, both, POPMUSIC as well as AMP may serve as a general frame encompassing various other approaches.

1.2.3 Miscellaneous

Target analysis may be viewed as a general learning approach. Given a problem, we first explore a set of sample instances and an extensive effort is made to obtain a solution which is optimal or close to optimality. The best solutions obtained provide *targets* to be sought within the next part of the approach. For instance, a TS algorithm may be used to bias the search trajectory toward already known solutions (or as close to them as possible). This may give some information on how to choose parameters for other problem instances.

A different acronym in this context is *path relinking* (PR) which provides a useful means of intensification and diversification. Here new solutions are generated by exploring search trajectories that combine elite solutions, i.e., solutions that have proved to be better than others throughout the search. For references on target analysis and PR see, e.g., [58].

Considering local search based on *data perturbation*, the acronym *noising method* may be related to the following approach, too. Given an initial feasible solution, the method performs some data perturbation [111] in order to change the values taken by the objective function of a problem to be solved. On the perturbed data a local search may be performed (e.g., following a SD approach). The amount of data perturbation (the noise added) is successively reduced until it reaches zero. The noising method is applied, e.g., in [23] for the clique partitioning problem.

The key issue in designing *parallel algorithms* is to decompose the execution of the various ingredients of a procedure into processes executable by parallel processors. Opposite to ant systems or GAs, metaheuristics like TS or SA, at first glance, have an intrinsic sequential nature due to the idea of performing the neighborhood search from one solution to the next. However, some effort has been undertaken to define templates for parallel local search (see, e.g., [119, 117, 26, 116]). A comprehensive treatment with successful applications is provided in [6]. The discussion of parallel metaheuristics has also led to interesting hybrids such as the combination of a population of individual processes, agents, in a cooperative and competitive nature (see, e.g., the discussion of *memetic algorithms* in [87]) with TS.

Neural networks may be considered as metaheuristics, although we have not considered them here; see, e.g., [108] for a comprehensive survey on these techniques for combinatorial optimization. On the contrary, one may use metaheuristics to speed up the learning process regarding artificial neural networks; see [7] for a comprehensive consideration.

Require: $\mathbf{x}^k$ incumbent solution; Ω^k current set of heuristic parameters
Ensure: $\mathbf{x}^{k+1}$ next solution
1. $\mathcal{N}(\mathbf{x}^k) \leftarrow$ `neighborhood_definition`$(\mathbf{x}^k, \Omega^k)$
2. $\mathbf{x}^{k+1} \leftarrow$ `neighborhood_exploration`$(\mathcal{N}(\mathbf{x}^k))$
3. $\Omega^{k+1} \leftarrow$ `parameters_update()`

Fig. 1.1 General Metaheuristic Iteration.

Stochastic local search (SLS) is pretty much all we know about local search but enhanced by randomizing choices. That is, an SLS algorithm is a local search algorithm making use of randomized choices in generating or selecting candidate solutions for given instances of optimization problems. Randomness may be used for search initialization as well as the computation of search steps. A comprehensive treatment of SLS is given in [75].

Furthermore, recent efforts on problems with multiple objectives and corresponding metaheuristic approaches can be found in [78, 41]. See, e.g., [105] for some ideas regarding GAs and fuzzy multi-objective optimization.

1.3 A Taxonomy

In this section, we present a taxonomy of metaheuristics along a single dimension of analysis. The driving factor is the way in which the neighborhood is defined with respect to each metaheuristic approach. Alternative classifications have been proposed, e.g., in [19, 64].

From a general perspective, each metaheuristic paradigm can be seen as made up by three major ingredients, which are repeatedly used at each iteration until specified stopping criteria are reached. A generalization of a single iteration of a metaheuristic scheme is given in Figure 1.1.

As illustrated in Step 1 of Figure 1.1, a common ingredient of each metaheuristic paradigm is the existence of a rule aimed at iteratively guiding the search trajectory, i.e., a set of rules to define a neighborhood. In turn, such neighborhood demarcates which solutions can be reached starting from the incumbent solution. In line with this observation, a possible dimension along which a taxonomy of metaheuristics can be created is given by the way in which neighborhoods are built. A classification of metaheuristics along this dimension leads to the definition of at least two broad classes:

- *model-based heuristics*: as in [133], with this term we refer to metaheuristic schemes where new solutions are generated by using a model. Consequently, the neighborhood is implicitly defined by a set of parameters, and iteratively updated during the search process;

- *method-based heuristics*: as in [109], with this term we make reference to heuristic paradigms in which new solutions are sought in a neighborhood whose structure is dictated by the method used to explore the neighborhood itself. Consequently, the neighborhood is implicitly defined by the predetermined method employed.

By observing the underlying philosophy of these two broad classes, a clear dichotomy arises: on the one hand, model-based heuristics tackle the original optimization problem by defining and iteratively updating a model aimed at identifying prominent features in good solutions and at replicating these features in future solutions. Consequently, what determines whether a point belongs to the neighborhood is the set of parameters that defines the model and the 'probability' of generating such point under the current model. On the other hand, method-based heuristics are driven by the technique used to solve a "reduced" version of the original problem, i.e., a problem in which only a subset of the original solution space is considered. Consequently, in method-based heuristics what dictates the structure and shape of the neighborhood is the optimization method employed to explore the neighborhood itself, whether it be a classical mathematical programming technique, e.g., branch and bound, dynamic programming, etc., or a simple enumeration-based technique.

Model-based heuristics are generally based upon the identification of a set of parameters, defining a model that, in turn, well captures some features of the search space. This type of heuristics heavily relies on a set of update schemes, used to progressively modify the model itself in such a way that, after each update, the possibility of obtaining higher quality solutions under the new model is increased. Consequently, in Step 1 of the general `Metaheuristic_Iteration()`, all the solutions that "comply" with the requirements enforced by the model upon the search space are included in the current neighborhood. A special role is played by Step 3 of the same algorithm, in which the parameters of the model are updated via the application of learning mechanisms. In this phase, modifications are applied to the model and/or its parameters to reflect insight collected and generated during the search phase.

A well-known paradigm that can be interpreted under the philosophy of the model-based method is the CE, where a stochastic model is continually updated to reflect the findings of the last iteration of the search process. Other metaheuristics that can be seen as model-based are ACO (as well as other methods belonging to the *Swarm Intelligence* field), where a construction graph and stochastic procedures called *ants* are employed; semi-greedy heuristics [70], including the greedy randomized adaptive search procedure *GRASP* [43], where the greedy function that guides the selection of the best candidates defines a stochastic model; and GAs, where adaptive search procedures based upon genetics are put into place. The GA paradigm heavily relies on a model, defined by a set of operators, that determines which solutions will be included in the next generation, i.e., in the current neighborhood.

More generally, evolutionary algorithms could similarly be included into the category of model-based metaheuristics.

On the other side of the spectrum we find metaheuristic paradigms driven by a method, rather than by a model. The basic ingredient of such an approach is the existence of a search method, whether it be an exact method or a heuristic method, that is used to explore the neighborhood itself. Consequently, the size and cardinality of the neighborhood depend upon the ability of the method itself to explore the portion of the search space included in the neighborhood. Within the class of method-based metaheuristics we can introduce a further level of classification, according to the nature of the method employed to explore the neighborhood. Broadly speaking, method-based heuristics could be divided into two categories, those for which classical mathematical programming techniques are employed to explore the neighborhood and those for which enumeration-based techniques are used to conduct the exploration of the basin of solutions.

A cardinal concept for the method-based heuristics is connected to the introduction of a "distance" metric. A distance is used to draw the boundaries of the neighborhood around the incumbent solution, in such a way that only points whose distance from the incumbent solution is within a threshold are included in the neighborhood itself. Consequently, the neighborhood is explicitly defined by the notion of distance adopted. On the other hand, the definition of the threshold distance is strongly connected with the capabilities of the method used to explore the neighborhood. In other words, the cardinality of the neighborhood is chosen in such a way that, on the one hand, it will be large enough to have a reasonable chance of containing a solution better than the incumbent one and, on the other hand, small enough to be explored by employing the method at hand in a reasonable amount of computational time.

Historically, the first metaheuristics developed might be regarded as belonging to this class, e.g., SA or TS. Let us consider, *e.g.*, the TS metaheuristic. Given an incumbent solution $\mathbf{x}^i$, a distance function $d(\mathbf{x}_1, \mathbf{x}_2)$ and a threshold value δ, only solutions for which $d(\mathbf{x}^i, \mathbf{x}) \leq \delta$ will be included into the current neighborhood. Once the neighborhood definition phase is terminated, a method capable of exploring such neighborhood in a reasonable amount of computational time is employed to find a possibly better solution. Consequently, while the neighborhood is explicitly defined by the value of δ, it is possible to say that such neighborhood is implicitly defined by the method used, since the value of δ depends upon the capabilities of the method itself.

It is worth noting that these metaheuristics can, and in general, do use a set of parameters to refine the definition of the neighborhood. For example, let us once more consider the case of the TS metaheuristic. The neighborhood is first defined according to a "distance" from the incumbent solution and, then, refined via the application of, *e.g.*, the tabu list, with the effect of eliminating some of the solutions from the current neighborhood. However, it should be

evident that the major ingredient used in determining the neighborhood is still related to the concept of distance from the incumbent solution.

Other metaheuristics that fit into this category are VNS [68] and the *pilot method*. In VNS, the neighborhood is altered during the search in such a way that increasingly distant neighborhoods of a given solution are explored. However, a "method", e.g., the local search routine, must be applied to perform the search over the neighborhood. Thus, the way in which neighborhoods are built is influenced by the method used to explore the portion of the search space at hand. Similarly, in the pilot method, the core idea is that the neighborhood is defined by a look-ahead mechanism. Consequently, there is a method (even a simple local search) that determines the shape, or the deepness, of the neighborhood itself.

More recently, metaheuristics employing classical mathematical programming techniques to explore the neighborhood have been proposed. Let us consider the case of the CM. After defining a corridor around the incumbent solution, an optimization method is then applied within the neighborhood defined by the corridor with the aim of finding an improved solution. Consequently, in Step 1 of the general metaheuristic iteration of Figure 1.1, only solutions within the corridor, or within a predefined distance from the incumbent, will be included in the neighborhood. As previously mentioned, an optional feature of the `neighborhood_definition()` phase is the application of a set of criteria to refine the neighborhood, e.g., a tabu list, aspiration criteria, etc. Step 2 of the algorithm relies on the use of either an enumeration-based technique, or a classical mathematical programming technique (branch and bound, dynamic programming, etc.) to explore the neighborhood. Finally, Step 3 of the algorithm, `parameters_update()` can include, e.g., the dynamic update of distance and corridor parameters, depending upon the current status of the search.

Beside the corridor method, other concepts that fall into this category are, e.g., constraint programming [103], in which the corridor is defined by the subsequent application of constraints and conditions to be satisfied by the solution, and local branching [47], in which the neighborhood is defined by introducing linear inequalities, called local branching cuts. Once a restricted neighborhood is so defined, an exact technique, i.e., linear programming, is used in the spirit of the branch and bound framework.

A further example which might be interpreted as a method-based technique is the POPMUSIC framework [112], where one wants to solve, preferably to optimality, smaller portions of the solution space, based upon an available feasible solution.

More recently, the *relaxation induced neighborhood search* method (RINS) has been introduced [27]. The RINS defines a neighborhood exploiting information contained in the linear programming (LP) relaxation and can naturally be seen as a method-based framework for mixed integer programs (MIP). The central idea of the method is related to the exploitation of a "relaxed" solution to define a core problem, smaller than the original MIP. The core

problem is identified at any given node of the branch and cut tree, by first fixing all variables that have the same values both in the incumbent (feasible) solution and the relaxed solution. Next, a branch-and-cut framework is used to solve to optimality the reduced problem on the remaining variables, called sub-MIP. The process is iteratively applied at any node of the global tree, since the LP relaxation induced solution is different and, therefore, gives raise to different sub-MIPs.

1.4 Hybrids with Exact Methods

In recent years, a lot of attention has been devoted to the integration, or hybridization, of metaheuristics with exact methods (see, e.g., [95, 96] for a survey and a taxonomy about hybrid approaches in combinatorial optimization, respectively.) In this section, we use the term *hybrid* in a somehow restrictive way, since we classify as *hybrid approaches* only those approaches that combine the use of exact techniques with metaheuristic frameworks. Consequently, algorithms that combine together different metaheuristics are not included in this analysis although they could be and are also termed hybrid.

This exposition also relates to the term *Matheuristics*, which describes works that also are along these lines, e.g., exploiting mathematical programming techniques in (meta)heuristic frameworks or on granting to mathematical programming approaches the cross-problem robustness and constrained-CPU-time effectiveness which characterize metaheuristics. Discriminating landmark is some form of exploitation of the mathematical formulation of the problems of interest [67].

Generally speaking, hybrid algorithms present a so-called "master-slave" structure of a guiding process and an application process. Either (i) the metaheuristic acts at a higher level and controls the calls to the exact approach, or (ii) the exact technique acts as the master and calls and controls the use of the metaheuristic scheme.

Hybrid algorithms of type (i) are such that the *definition* of the neighborhood follows the logic of a metaheuristic, while the *exploration* of the neighborhood itself is left to the exact approach. From this perspective, the metaheuristic acts as the master by defining the size and boundaries of the neighborhood and by controlling repeated calls to the exact method, which, in turn, acts as an application process, by exploring each neighborhood in an exact fashion. Algorithms that fall into this category are, e.g., those inspired by the CM, in which large scale neighborhoods are searched exhaustively through an exact method applied on a sub-portion of the search space. The call to the exact method is managed by a scheme that heuristically defines a corridor around an incumbent solution (cf. the previous section).

A similar philosophy is shared by the *large scale neighborhood search* (see, e.g., [5] for a survey), in which exponentially large neighborhoods are searched to optimality by means of, e.g., ad-hoc enumeration schemes, dynamic programming schemes, etc.

Along the same line, the RINS as well as local branching could be seen as algorithms of type (i), at least in *spirit*. For example, even though the RINS is casted into a branch and cut approach, and, therefore, the guiding process is an exact approach, the logic of the method is centered upon metaheuristic-type features, such as neighborhood definition, diversification and intensification. It is worth noting, though, that in these two approaches, no real metaheuristic is ever deployed, since they entirely rely on the branch and bound framework. However, they can still be seen as hybrid approaches because of the embedded metaheuristic philosophy that drives the search process.

On the other hand, we also have hybrid approaches of type (ii), in which the metaheuristic scheme is embedded into the solver. Modern branch and cut solvers exploit the potentials of (meta)heuristics to quickly get good quality solutions, especially at early stages of the tree exploration. Related bounds are then employed to prune branches of the tree and, consequently, contribute to speed up the search process and to reduce the overall computational effort. Since it has been observed that in some important practical cases MIP solvers spend a large amount of computational time before finding the first feasible solution, [46] introduced a heuristic scheme, further improved in [3, 16], called the *feasibility pump*, aimed at quickly finding good quality initial solutions. Such initial solutions are obtained via a sequence of roundings, based upon continuous relaxation solutions, that converge to a feasible MIP solution. Clearly, such heuristic-type schemes can also be used to quickly find initial solutions to be fed to type (i) hybrid algorithms, such as the CM, the RINS as well as local branching (see also [56, 57, 40] for heuristic methods for MIP feasible solution generation).

In a fashion similar to hybrid approaches of type (ii), some researchers have also employed metaheuristic schemes for column generation and cut generation within branch and price and branch and cut frameworks, respectively (see, e.g., [44] and [94]). In addition, one may investigate hybrids of branch and bound and metaheuristics, e.g., for deciding upon branching variables or search paths to be followed within a branch and bound tree (see, e.g., [130] for an application of reactive TS). Here we may also use the term *cooperative solver*.

A key question that arises when designing a hybrid algorithm concerns which components should be "hybridized" to create an effective algorithm. While providing an all-encompassing rule for hybridization does not seem to be a feasible approach, from the analysis of the state of the art of hybrid algorithms some interesting guidelines emerge.

A method-based approach is centered upon the exploitation of an effective "method" to solve the problem at hand. Consequently, the starting point lies

in the identification of the most effective(s) method(s) with respect to the optimization problem. For example, the design of a CM inspired algorithm requires previous knowledge about which method could effectively tackle the problem if this were of reduced size. Thus, the identification of the method to be used constitutes the central point in the design of the algorithm itself.

A basic ingredient of a method-based algorithm concerns the heuristic rule used to draw the boundaries of the neighborhood upon which the method will be applied, which is the *design of the neighborhood* itself in terms of how large the neighborhood should be. Size and boundaries of such neighborhood depend on the "power" of the method used, i.e., on its ability to explore large portions of the solution space in a reasonable amount of computational time. While determining the appropriate dimension of the neighborhood for the method at hand is an issue of algorithm fine tuning (as presented in Section 1.6), some general considerations are related to the fitness landscape analysis as well as the computational complexity of the method itself. Roughly speaking, given an optimization method and its worst case computational complexity in terms of size of the input, it is possible to determine the maximum size of the neighborhood that guarantees running times below a desired threshold. On the other hand, since computational complexity analysis mainly deals with worst case scenarios, it seems beneficial to employ fitness landscape analysis techniques (e.g., connectivity measures) to draw tightest complexity bounds that translate directly into larger neighborhoods.

Another guideline is provided by the *intensification-diversification trade-off*. By reviewing hybrid algorithms proposed in the literature, many times it is possible to identify a predominant focus, in the sense that some algorithms put a higher emphasis on diversification of solutions, while others emphasize the intensification of the search in promising regions. For example, as illustrated in [47], the application of valid inequalities in the spirit of local branching fosters the intensification of the search in a given neighborhood, hence allowing to find good quality solutions early on in the search process. In a similar fashion, the CM seems to put more emphasis on intensifying the search within a promising region, without defining specific restarting mechanisms to achieve diversification. On the other hand, a method such as the RINS, based upon a solution of the LP relaxation of the MIP, puts more emphasis on diversification, since at each node of the search tree a different LP induced solution is used and, consequently, different solutions feasible with respect to the MIP will be produced.

Finally, an interesting line of research aimed at grasping a clearer understanding of why some search techniques are successful on a given problem class is related to the *fitness landscape analysis*. As mentioned in [18, 115], a central measure of landscape structure is the fitness-distance correlation, which captures the correlation between objective function value and the length of a path to an optimal solution within the fitness landscape. Such a measure is used to explain why local search techniques perform well in tackling certain problems. However, the link between problem difficulty and

fitness landscape is, as of today, not completely understood (see also the idea of target analysis mentioned above).

Fitness landscape analysis can be used with at least two goals in mind:

- on the one hand, as brought out in [126], this kind of analysis helps to understand what makes a problem hard or, conversely, well suited, for a specific search technique. Information such as fitness-distance correlation, ruggedness, nodes connectivity, and drifting can be exploited to design an effective metaheuristic scheme as well as to identify which components should be hybridized;
- on the other hand, as illustrated in [24], fitness landscape analysis can help to identify which formulation of the same problem will be more suitable with respect to an available algorithm. For example, [24] were able to "predict" the behavior of a VNS algorithm upon two different formulations of the Golomb Ruler problem and, consequently, to select the formulation that better fitted with the potentials of their algorithm.

While this field of study seems promising in grasping a better understanding of the "hows" and "whys" of metaheuristics, an important issue of generalization of results has already been noticed. As mentioned in [126], the results obtained so far have mainly been used *a posteriori*, to justify the use of a given algorithm and its features. However, it is unclear how this kind of analysis can be extended to develop improved algorithms, since there is no clear understanding about the general validity of the findings of the proposed models. However, it is worth noting that, from the methodological perspective, the contribution of the fitness landscape analysis is far from negligible, since its focus is well oriented toward interpreting and partially explaining successes and failures of metaheuristic-based algorithms.

1.5 General Frames: A Pool-Template

An important avenue of metaheuristics research refers to general frames (e.g., to explain the behavior and the relationship between various methods) as well as the development of software systems incorporating metaheuristics (eventually in combination with other methods). Besides other aspects, this takes into consideration that in metaheuristics it has very often been appropriate to incorporate a certain means of diversification versus intensification to lead the search into new regions of the search space. This requires a meaningful mechanism to detect situations when the search might be trapped in a certain area of the solution space. Therefore, within intelligent search the exploration of memory plays a most important role.

In [64] a *pool template* (PT) is proposed as can be seen in Figure 1.2. The following notation is used. A pool of $p \geq 1$ solutions is denoted by P. Its input and output transfer is managed by two functions which are called *IF*

```
1.  Initialize P by an external procedure
WHILE termination=FALSE DO BEGIN
    2.  S := OF(P)
    3.  IF s > 1 THEN S' := SCM(S) ELSE S' := S
    4.  S'' := IM(S')
    5.  P := IF(S'')
END
6.  Apply a post-optimizing procedure to P
```

Fig. 1.2 Pool Template.

and OF, respectively. S is a set of solutions with cardinality $s \geq 1$. A solution combination method (procedure SCM) constructs a solution from a given set S, and IM is an improvement method.

Depending on the method used, in Step 1 either a pool is completely (or partially) built by a (randomized) diversification generator or filled with a single solution which has been provided, e.g., by a simple greedy approach. Note that a crucial parameter that deserves careful elaboration is the cardinality p of the pool. The main loop, executed until a termination criterion holds, consists of Steps 2–5. Step 2 is the call of the output function which selects a set of solutions, S, from the pool. Depending on the kind of method represented in the PT, these solutions may be assembled (Step 3) to a (set of) working solution(s) S' which is the starting point for the improvement phase of Step 4. The outcome of the improvement phase, S'', is then evaluated by means of the input function which possibly feeds the new solution into the pool. Note that a post-optimizing procedure in Step 6 is for facultative use. It may be a straightforward greedy improvement procedure if used for single-solution heuristics or a pool method on its own. As an example we quote a *sequential* pool method, the TS with PR in [11]. Here a PR phase is added *after* the pool has been initialized by a TS. A *parallel* pool method on the other hand uses a pool of solutions *while* it is constructed by the guiding process (e.g., a GA or scatter search).

Several heuristic and metaheuristic paradigms, whether they are obviously pool-oriented or not, can be summarized under the common PT frame. We provide the following examples:

a) Local Search/SD: PT with $p = s = 1$.
b) SA: $p = 2, s = 1$ incorporating its probabilistic acceptance criterion in IM. (It should be noted that $p = 2$ and $s = 1$ seems to be unusual at first glance. For SA we always have a current solution in the pool for which one or more neighbors are evaluated and eventually a neighbor is found which replaces the current solution. Furthermore, at all iterations throughout the search the so far best solution is stored, too (even if no real interaction between those two stored solutions takes place). The same is also valid for

a simple TS. As for local search the current solution corresponds to the best solution of the specific search, we have $p = 1$.)

c) Standard TS: $p = 2, s = 1$ incorporating adaptive memory in *IM*.

d) GAs: $p > 1$ and $s > 1$ with population mechanism (crossover, reproduction and mutation) in *SCM* of Step 3 and without the use of Step 4. (One might argue that $p \geq 1$ is also possible.)

e) Scatter Search: $p > 1$ and $s > 1$ with subset generation in *OF* of Step 2, linear combination of elite solutions by means of the *SCM* in Step 3, e.g., a TS for procedure *IM* and a reference set update method in *IF* of Step 5.

f) PR (as a parallel pool method): $p > 1$ and $s = 2$ with a PR neighborhood in the *SCM*. Facultative use of Step 4.

g) CE: $p > 1$ and $s = \rho p$, with $\rho \in (0, 1)$, where the *SCM* is used to update the underlying stochastic model by capturing features of solutions in S and without the use of Step 4.

1.6 Fine Tuning and Evaluation of Algorithms

In this section, we discuss two important issues related to the design and analysis of metaheuristics and related algorithms. On the one hand, as mentioned in the introduction, one key factor that has a bearing on the overall performance of most of these algorithms is the calibration of the algorithmic parameters. Thus, a natural question is about how to select an appropriate set of values for these parameters. This important topic in metaheuristic design goes under the name of *fine tuning* of algorithmic parameters.

A second relevant issue, related to the analysis of an algorithm, is concerned with the empirical evaluation of the performance of the algorithm itself. Due to the stochastic nature of many metaheuristic schemes, a problem of reproducibility of results arises. Many researchers advocate the definition of a set of standards to increase the objectivity of the "ranking" and evaluation of metaheuristics. However, the literature does not seem to be mature with respect to this topic.

1.6.1 Fine Tuning of Metaheuristics

According to [4], there is evidence that 10% of the time required to develop a new metaheuristic is devoted to the actual development and that the remaining 90% is spent on fine tuning of algorithmic parameters. In addition, fine tuning of parameters strongly affects the final performance of an algorithm. For these reasons, it is of paramount importance to make a concerted effort in identifying and establishing a set of "standard" techniques to fine-tune a metaheuristic. One of the major achievements of such an effort would be to

offset parameter specific issues in evaluating an algorithm. In addition, reproducibility of results would also be enhanced by such an approach, by making transparent the way in which parameter values should be set to tackle a given problem instance.

In the literature, some attempts to use statistically robust methods have been presented. For example, in [4], a tool called CALIBRA is proposed as a procedure that finds good values for up to five algorithmic parameters. They exploit Taguchi fractional factorial design to reduce the overall number of trials required to train the model. Interestingly, the authors tested CALIBRA on a set of six different algorithms and use hypothesis testing to assert whether the parameter values suggested by the tool allow to find solutions which are significantly better than those proposed by the original authors of the algorithms.

In [25], a four-step procedure is proposed, in which a two-level factorial design is coupled with linear regression to find a linear approximation of the response surface of the set of parameters. Subsequently, a gradient descent technique is employed to find a "good" value for each parameter. Finally, the method is applied on a set of benchmark problems, and results are collected to show that a good parameter setting leads to improvements in the objective function value. A similar technique has been employed in [21], where a circumscribed central composite design is used to generate observations and a higher degree polynomial is then used to approximate the response surface of a set of parameters. Once the response surface is known, a global optimization method is employed to find the optimal parameter value with respect to the surface. The model is built on a set of training instances and then validated on a different set of testing instances of some lot sizing problems.

In [89], a nonlinear response surface is used to capture the impact of parameters on a SA algorithm. A good set of values of such parameters is then determined via a modified simplex method for nonlinear programming, that can be used to deal with bounds on parameter values. The proposed method is tested on three different combinatorial optimization problems and the collected results are then compared with those of other SA implementations, whose fine tuning was achieved via extensive experiments. The results show that there is no statistical difference in performance between the two types of SA algorithms and, consequently, that the proposed approach can be used to conduct an "automatic" fine tuning. A similar approach was followed in [90], where a full-factorial design was employed to define a response surface for up to four parameters of a GA. The fine-tuned algorithm was finally tested on a pool of DNA sequence assembly problems and results were collected to illustrate the effectiveness of the procedure.

A somehow different approach is proposed in [131], where two statistical tests are employed to fine-tune a set of algorithmic parameters of a TS algorithm and to validate the proposed statistical model. First, a Friedman test is used to detect significance of a specific parameter upon the algorithm performance and, via a series of pairwise comparisons, to identify a good pa-

rameter value; subsequently, a Wilcoxon test is employed to verify whether there is a statistically significant difference between any two F-runs of the algorithm. The TS algorithm is then used upon a set of problems drawn from the telecommunication network design field.

A relevant issue in fine tuning methods is related to the "budget" available to carry on the experiments, in the sense that the type of techniques used depends on the amount of time and computational power available. These elements affect the number of factors (parameters) and the number of levels (values) that can be considered in the experimental design. In this regard, as illustrated in [4], the use of a "fractional factorial design" can help in reducing the number of runs required to collect results.

A second issue concerns the analysis of results in terms of robustness and sensitivity. In many circumstances, it is necessary to provide not only a "good" set of parameter values but also a measure of robustness and sensitivity with respect to those parameters. For this reason, the experiment should be designed in such a way that training and testing sets are used to create and to validate the model, respectively.

Finally, an important issue that needs to be addressed when fine tuning algorithmic parameters is the ultimate goal that one wants to achieve. In the literature, many times it is implicitly assumed that the goal is to maximize solution quality and, with this in mind, one wants to find a good set of parameters. However, alternative goals could also be desirable, such as, e.g., minimizing computational time to a target solution, generating a pool of solutions with maximum "diversification" [63], obtaining good quality solutions early on in the search process, as well as a weighted combination of such goals. Consequently, the fine tuning process should be designed in such a way that a trade-off between conflicting goals could be achieved.

In the spirit of what we have seen in the reactive TS fine tuning also concerns means of autoadaptivity of parameter settings. In a more general setting an excellent recent treatment is provided in [14].

1.6.2 Empirical Evaluation of Metaheuristics

As stated in [10], it is important to "promote thoughtful, well-planned, and extensive testing of heuristics, full disclosure of experimental conditions, and integrity in and reproducibility of the reported results."

With this aim in mind, a number of interesting issues have been proposed with respect to how to measure the performance of a metaheuristic. For example, some relevant issues connected with the evaluation of metaheuristics are:

- computational experiment design, with a clear objective in mind (e.g., to illustrate solution quality performance, robustness, quality versus time, etc.);

- testing on benchmark instances, if available, or with respect to (dual) gaps if possible. Results should be reported with measures of variability and robustness;
- if no comparison with other algorithms from the literature is possible, the algorithm should at least be compared with a simple random restart procedure;
- identification of the contribution of each major feature of the algorithm on the final performance, to detect which features are relevant in achieving the declared level of performance;
- measurement of statistically significant differences among algorithms, with the aim of *ranking* a pool of algorithms designed to tackle the same class of problems.

In this section, we focus on two major issues: *reproducibility* and *ranking* of an algorithm. On the one hand, the experimental results should be reported in such a way that reproducibility is ensured while, on the other hand, a statistically sound comparison of the proposed method with other approaches from the literature should be presented with the aim of detecting meaningful improvements in the state of the art.

The issue of reproducibility has been addressed in [10]. In order to be of any scientific value, the experiments used to assess an algorithm in general, and a metaheuristic in particular, should be entirely reproducible by others. On the one hand, documentation will help in fostering reproducibility. For this reason, not only the algorithmic steps but also specifics about the implementation, the data structure employed, the parameter settings, the random number process generation (if applicable), etc., should be provided. In addition, making available to the community the source code as well as the instances used fosters reproducibility and enhances the quality of the scientific work.

Many statistical tests have been proposed to determine whether one algorithm outperforms another. The way in which this issue should be addressed is influenced by a number of relevant factors, such as, e.g., the nature of the algorithm itself (deterministic or stochastic), the size of the available sample (test bed), and, of course, the measure(s) used to draw conclusions about the quality of a given algorithm. In [31], a taxonomy of statistical questions, applied to learning algorithms, is presented and some hints about how to address the issue of comparing algorithms with respect to each category are given.

In the operations research field, the issue of comparing two algorithms with each other arises in at least two different contests:

- on the one hand, one might want to compare two different versions of the same algorithm using different parameter settings to *fine tune* the algorithm itself. In this case, the practitioner aims at detecting whether a given set of parameters produces better performance;

- on the other hand, to *validate* an algorithm, a researcher will have to compare the performance of the proposed algorithm against, at least, those of the best algorithm available for a specific class of problems. In this case, the underlying assumption, quite common in the operations research field, is that the algorithm is designed to perform in a specific domain, i.e., to tackle a single class of problems, and that it is possible to identify a state of the art algorithm.

While the literature on the use of statistical analysis for hypothesis testing is abundant (e.g., [132], [50]), in general, only rudimentary techniques are used to present results and assert the quality and soundness of the same results produced by a new metaheuristic. Other fields seem more mature when it comes to statistical validation of results. For example, the machine learning community has become increasingly aware of the importance of validating results with sound statistical analysis [29]. In [31], five statistical tests for comparison of two different algorithms are presented. These tests are experimentally compared with respect to the probability of incurring Type I errors (incorrectly detecting a difference when no difference exists). More sophisticated tests are proposed in [29], which can be used to compare multiple algorithms over multiple data sets.

In the field of metaheuristics, the use of statistical tests is somehow present when fine tuning algorithms. For example, as mentioned in Section 1.6.1, [4], [131], [89], use different statistical tests to assert the quality of the fine tuning technique. However, the literature concerning authors that employ statistical analysis to *compare* two, or a set, of metaheuristics upon a given set of instances is quite scanty. A good introduction to the topic can be found in [113]. In this paper, only the case in which two algorithms are to be compared is presented. A number of statistical tests is reviewed, both parametric and non-parametric tests, and a new non-parametric test is proposed (see also [110] for some ideas).

The major issue in identifying which test should be used to compare two algorithms is the identification of the key factor to be used to "judge" such algorithms. For example, in the contest of optimization, a common measure of quality is the objective function value. Consequently, one might want to compare two algorithms, say algorithm A and algorithm B, in terms of a set of related objective function values, say z^A and z^B. The null hypothesis is then defined as $H_0 : z^A - z^B = 0$ and a one-sided or a two-sided test is designed. In general, classical parametric approaches are used to test such hypothesis such as, e.g., a (paired) t-test, which checks whether the average difference in performance over the data set is significantly different from zero. However, as brought out by many authors (see, e.g., [29]), the t-test suffers from a number of weaknesses, mainly that the underlying assumption of normal distributions of the differences between the two random variables could not be realistic, especially when the number of benchmark instances available is not large.

An alternative approach relies on the use of non-parametric tests for comparing proportions. Let us suppose that we count a success for algorithm A

every time algorithm A outperforms algorithm B on a given instance (e.g., in terms of objective function value, running time, etc.). The researcher is interested in estimating the success probabilities of the two algorithms, say p_A and p_B. After estimating empirically such probabilities, a non-parametric test (e.g., McNemar test, Fisher exact test) could be used to "rank" such algorithms in terms of effectiveness, indicating which algorithm is more successful on the same data set. A non-parametric test that can be used to "rank" algorithms based upon the success probabilities is proposed by [113]. In addition, such test overcomes one important limitation of the McNemar test, namely the fact that these tests require pairwise comparisons. Many times in the field of metaheuristics researchers test their algorithms on randomly generated instances, whose generation process is described while the actual instances are not made available to the community. Hence, it is not always possible to compare two algorithms on the same set of instances and the McNemar test might not be significant. The test proposed in [113] overcomes such obstacle by taking into account the total number of runs of algorithm A and of algorithm B (which might be different) in comparing proportions and proves to be more powerful than the Fisher test.

More complicated scenarios could be envisioned when more than one dimension of evaluation is taken into account. For example, quite often a trade off between solution quality and computational time arises. Consequently, one might be interested in evaluating, or ranking, algorithms along these two conflicting dimensions. In addition, if the metaheuristic is part of a broader hybrid algorithm, such as those mentioned in Section 1.4, a valid measure of effectiveness could be the "degree of diversification" of the solutions provided by the metaheuristic. Thus, one might want to evaluate two algorithms according to their ability of generating diversified solutions, which are, then, fed to a global optimizer. In these cases, where one wants to evaluate the effect of a pool of factors, a multi-objective problem could be defined, in such a way that a "weight" is assigned to each criterion and the "success" of an algorithm is measured as the weighted sum of the different criteria.

Finally, one last word could be spent about which algorithm(s) the proposed metaheuristic should be compared against. Generally speaking, the goal is to compare the new metaheuristic with established techniques, e.g., the best algorithm available in the literature. However, as pointed out in [10], rather than reporting comparisons with results produced on different machines and different instances, it is better to obtain or, in its defect, to recode, existing algorithms and conduct a fair comparison of the two algorithms on the same set of instances and the same machine. On the other hand, if no algorithms have been proposed for the problem at hand, a more general method, e.g., one based on linear programming or integer programming, could be used to obtain bounds on the objective function values. Whenever a stochastic scheme is proposed, at least a comparison with a simple random restart procedure should be carried on, to show that the proposed algorithm

performs *significantly better* (in a statistical sense) than the random restart procedure.

1.7 Optimization Software Libraries

Besides some well-known approaches for reusable software in the field of exact optimization (e.g., CPLEX[2] or ABACUS[3]) some ready-to-use and well-documented component libraries in the field of local search based heuristics and metaheuristics have been developed; see the contributions in [125].

The most successful approaches documented in the literature are the Heuristic OpTimization FRAMEwork HotFrame of [45] and EASYLOCAL++ of [30]. HotFrame, as an example, is implemented in C++, which provides adaptable components incorporating different metaheuristics and an architectural description of the collaboration among these components and problem-specific complements. Typical application-specific concepts are treated as objects or classes: problems, solutions, neighbors, solution attributes and move attributes. On the other side, metaheuristic concepts such as different methods described above and their building-blocks such as tabu criteria or diversification strategies are also treated as objects. HotFrame uses genericity as the primary mechanism to make these objects adaptable. That is, common behavior of metaheuristics is factored out and grouped in generic classes, applying static type variation. Metaheuristics template classes are parameterized by aspects such as solution spaces and neighborhood structures.

Another well-known optimization library is the COIN-OR library[4], an open-source suite for the optimization community. An effort in the development of standards and interfaces for the interoperability of software components has been put forth. Some classes that implement basic ingredients of metaheuristics, e.g., Tabu Search Project, have been developed. However, the development of general software frameworks for metaheuristic paradigms, although strategic in defining standards and commonalities among approaches, is still in its infancy.

1.8 Conclusions

Over the last decades metaheuristics have become a substantial part of the optimization stockroom with various applications in science and, even more

[2] `www.ilog.com`

[3] `www.informatik.uni-koeln.de/abacus`

[4] `http://www.coin-or.org`

important, in practice. Metaheuristics have become part of textbooks, e.g. in operations research, and a wealth of monographs (see, e.g., [118, 58, 86, 36]) is available. Most important in our view are general frames. Adaptive memory programming, Stochastic Local Search, an intelligent interplay of intensification and diversification (such as ideas from POPMUSIC), and the connection to powerful exact algorithms as subroutines for handable subproblems and other means of hybridization are avenues to be followed.

Applications of metaheuristics are almost uncountable and appear in various journals (e.g., *Journal of Heuristics*), books, and technical reports every day. A helpful source for a subset of successful applications may be special issues of journals or compilations such as [98, 124, 100, 34], just to mention some. Specialized conferences like the *Metaheuristics International Conference* (MIC) are devoted to the topic (see, e.g., [88, 124, 102, 101, 76, 34]) and even more general conferences reveal that metaheuristics have become part of necessary prerequisites for successfully solving optimization problems (see, e.g., [61]). Moreover, ready to use systems such as class libraries and frameworks have been developed, although usually restricted to be applied by the *knowledgeable* user.

Specialized applications also reveal research needs, e.g., in dynamic environments. One example refers to the application of metaheuristics for online optimization; see, e.g., [65].

From a theoretical point of view, the use of most metaheuristics has not yet been fully justified. While convergence results regarding solution quality exist for most metaheuristics once appropriate probabilistic assumptions are made (see, e.g., [66, 8, 42]), these turn out not to be very helpful in practice as usually a disproportionate computation time is required to achieve these results (usually convergence is achieved for the computation time tending to infinity, with a few exceptions, e.g., for the reverse elimination method within tabu search or the pilot method where optimality can be achieved with a finite, but exponential number of steps in the worst case). Furthermore, we have to admit that theoretically one may argue that none of the described metaheuristics is *on average* better than any other. Basically this leaves the choice of a best possible heuristic or related ingredients to the ingenuity of the user/researcher. Some researchers related the acronym of hyper heuristics to the question which (heuristic) method among a given set of methods to choose for a given problem; see, e.g., [20].

Moreover, despite the widespread success of various metaheuristics, researchers occasionally still have a poor understanding of many key theoretical aspects of these algorithms, including models of the high-level run-time dynamics and identification of search space features that influence problem difficulty. Moreover, fitness landscape evaluations are considered in its infancy, too.

From an empirical standpoint it would be most interesting to know which algorithms perform best under various criteria for different classes of problems. Unfortunately, this theme is out of reach as long as we do not have any

well accepted standards regarding the testing and comparison of different methods.

While most papers on metaheuristics claim to provide 'high quality' results based on some sort of measure, we still believe that there is a great deal of room for improvement in testing existing as well as new approaches from an empirical point of view (see, e.g., [10, 74, 83]). In a dynamic research process numerical results provide the basis for systematically developing efficient algorithms. The essential conclusions of finished research and development processes should always be substantiated (i.e., empirically and, if necessary, statistically proven) by numerical results based on an appropriate empirical test cycle. Furthermore, even when excellent numerical results are obtained, it may still be possible to compare with a simple random restart procedure and obtain better results in some cases; see, e.g., [62]. However, this comparison is often neglected.

Usually the ways of preparing, performing and presenting experiments and their results are significantly different. The failing of a generally accepted standard for testing and reporting on the testing, or at least a corresponding guideline for designing experiments, unfortunately implies the following observation: Parts of results can be used only in a restricted way, e.g., because relevant data are missing, wrong environmental settings are used, or simply results are glossed over. In the worst case non-sufficiently prepared experiments provide results that are unfit for further use, i.e., any generalized conclusion is out of reach. Future algorithm research needs to provide effective methods for analyzing the performance of, e.g., heuristics in a more scientifically founded way (see, e.g., [127, 4] for some steps into this direction).

A final aspect that deserves special consideration is to investigate the use of information within different metaheuristics. While the adaptive memory programming frame provides a very good entry into this area, this still provides an interesting opportunity to link artificial intelligence with operations research concepts.

References

1. E.H.L. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization*. Wiley, Chichester, 1997.
2. E.H.L. Aarts and M. Verhoeven. Local search. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 163–180. Wiley, Chichester, 1997.
3. T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4:77–86, 2007.
4. B. Adenso-Diaz and M. Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54:99–114, 2006.
5. R.K. Ahuja, O. Ergun, J.B. Orlin, and A.B. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102, 2002.
6. E. Alba, editor. *Parallel Metaheuristics*. Wiley, Hoboken, 2005.

7. E. Alba and R. Marti, editors. *Metaheuristic Procedures for Training Neural Networks*. Springer, New York, 2006.
8. I. Althöfer and K.-U. Koschnick. On the convergence of 'threshold accepting'. *Applied Mathematics and Optimization*, 24:183–195, 1991.
9. T. Bäck, D.B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing, Bristol, 1997.
10. R.S. Barr, B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1:9–32, 1995.
11. M.B. Bastos and C.C. Ribeiro. Reactive tabu search with path relinking for the Steiner problem in graphs. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 39–58. Kluwer, Boston, 2002.
12. R. Battiti. Machine learning methods for parameter tuning in heuristics. Position paper for the 5th DIMACS Challenge Workshop: Experimental Methodology Day, 1996.
13. R. Battiti. Reactive search: Toward self-tuning heuristics. In V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, editors, *Modern Heuristic Search Methods*, pages 61–83. Wiley, Chichester, 1996.
14. R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*. Springer, New York, 2009.
15. R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, pages 126–140, 1994.
16. L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed integer problems. *Discrete Optimization*, 4(1):77–86, 2007.
17. D.P. Bertsekas, J.N. Tsitsiklis, and C. Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3:245–262, 1997.
18. C. Bierwirth, D.C. Mattfeld, and J.P. Watson. Landscape regularity and random walks for the job-shop scheduling problem. In J. Gottlieb and G.R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization, 4th European Conference, EvoCOP 2004*, volume 3004 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 2004.
19. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35:268–308, 2003.
20. E.K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F.W. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 457–474. Kluwer, Boston, 2003.
21. M. Caserta and E. Quiñonez Rico. A cross entropy-lagrangean hybrid algorithm for the multi-item capacitated lot sizing problem with setup times. *Computers & Operations Research*, 36(2):530–548, 2009.
22. R. Cerulli, A. Fink, M. Gentili, and S. Voß. Extensions of the minimum labelling spanning tree problem. *Journal of Telecommunications and Information Technology*, 4/2006:39–45, 2006.
23. I. Charon and O. Hudry. The noising method: A new method for combinatorial optimization. *Operations Research Letters*, 14:133–137, 1993.
24. C. Cotta and A. Fernández. Analyzing fitness landscapes for the optimal golomb ruler problem. In G.R. Raidl and J. Gottlieb, editors, *Evolutionary Computation in Combinatorial Optimization, 5th European Conference, EvoCOP 2005*, volume 3448 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2005.
25. S. P. Coy, B.L. Golden, G.C. Rungen, and E.A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7:77–97, 2000.
26. T.G. Crainic, M. Toulouse, and M. Gendreau. Toward a taxonomy of parallel tabu search heuristics. *INFORMS Journal on Computing*, 9:61–72, 1997.
27. E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming A*, 102:71–90, 2005.

28. P. De Boer, D.P. Kroese, S. Mannor, and R.Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134:19–67, 2005.
29. J. Demšar. Statistical comparison of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
30. L. Di Gaspero and A. Schaerf. EASYLOCAL++: An object-oriented framework for the flexible design of local-search algorithms. *Software – Practice and Experience*, 33:733–765, 2003.
31. T.G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.
32. M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics*, B - 26:29–41, 1996.
33. M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, 2004.
34. K.F. Dörner, M. Gendreau, P. Greistorfer, W.J. Gutjahr, R.F. Hartl, and M. Reimann, editors. *Metaheuristics: Progress in Complex Systems Optimization*. Springer, New York, 2007.
35. K.A. Dowsland. Simulated annealing. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 20–69. Halsted, Blackwell, 1993.
36. J. Dreo, A. Petrowski, P. Siarry, and E. Taillard. *Metaheuristics for Hard Optimization*. Springer, Berlin, 2006.
37. G. Dueck and T. Scheuer. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.
38. C.W. Duin and S. Voß. Steiner tree heuristics - a survey. In H. Dyckhoff, U. Derigs, M. Salomon, and H.C. Tijms, editors, *Operations Research Proceedings 1993*, pages 485–496, Berlin, 1994. Springer.
39. C.W. Duin and S. Voß. The pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs. *Networks*, 34:181–191, 1999.
40. J. Eckstein and M. Nediak. Pivot, cut, and dive: a heuristic for 0-1 mixed integer programming. *Journal of Heuristics*, 13:471–503, 2007.
41. M. Ehrgott and X. Gandibleux. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research*, 34(9):2674–2694, 2007.
42. U. Faigle and W. Kern. Some convergence results for probabilistic tabu search. *ORSA Journal on Computing*, 4:32–37, 1992.
43. P. Festa and M.G.C. Resende. An annotated bibliography of GRASP. Technical report, AT&T Labs Research, 2004.
44. G.R. Filho and L.A. Lorena. Constructive genetic algorithm and column generation: an application to graph coloring. In *Proceedings of APORS 2000 - The Fifth Conference of the Association of Asian-Pacific Operations Research Society within IFORS 2000*.
45. A. Fink and S. Voß. HotFrame: A heuristic optimization framework. In S. Voß and D.L. Woodruff, editors, *Optimization Software Class Libraries*, pages 81–154. Kluwer, Boston, 2002.
46. M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, A 104:91–104, 2005.
47. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, B 98:23–47, 2003.
48. D.B. Fogel. On the philosophical differences between evolutionary algorithms and genetic algorithms. In D.B. Fogel and W. Atmar, editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 23–29. Evolutionary Programming Society, La Jolla, 1993.
49. D.B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, New York, 1995.
50. A. M. Glenberg. *Learning from Data: An Introduction to Statistical Reasoning*. Lawrence Erlbaum Associates, Mahwah, New Jersey, 1996.

51. F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.
52. F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13:533–549, 1986.
53. F. Glover. Tabu search – Part II. *ORSA Journal on Computing*, 2:4–32, 1990.
54. F. Glover. Scatter search and star-paths: beyond the genetic metaphor. *OR Spektrum*, 17:125–137, 1995.
55. F. Glover. Tabu search and adaptive memory programming – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*, pages 1–75. Kluwer, Boston, 1997.
56. F. Glover and M. Laguna. General purpose heuristics for integer programming - part I. *Journal of Heuristics*, 2(4):343–358, 1997.
57. F. Glover and M. Laguna. General purpose heuristics for integer programming - part II. *Journal of Heuristics*, 3(2):161–179, 1997.
58. F. Glover and M. Laguna. *Tabu Search*. Kluwer, Boston, 1997.
59. F.W. Glover and G.A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer, Boston, 2003.
60. D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, 1989.
61. B.L. Golden, S. Raghavan, and E.A. Wasil, editors. *The Next Wave in Computing, Optimization, and Decision Technologies*. Kluwer, Boston, 2005.
62. A.M. Gomes and J.F. Oliveira. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171:811–829, 2006.
63. P. Greistorfer, A. Lokketangen, D.L. Woodruff, and S. Voß. Sequential versus simultaneous maximization of objective and diversity. *Journal of Heuristics*, 14:613–625, 2008.
64. P. Greistorfer and S. Voß. Controlled pool maintenance for meta-heuristics. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution*, pages 387–424. 2005.
65. K. Gutenschwager, C. Niklaus, and S. Voß. Dispatching of an electric monorail system: Applying meta-heuristics to an online pickup and delivery problem. *Transportation Science*, 38:434–446, 2004.
66. B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13:311–329, 1988.
67. P. Hansen, V. Maniezzo, and S. Voß. Special issue on mathematical contributions to metaheuristics editorial. *Journal of Heuristics*, 15(3):197–199, 2009.
68. P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voß, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, Boston, 1999.
69. P. Hansen, N. Mladenović, and D. Perez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.
70. J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.
71. A. Hertz and D. Kobler. A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, 126:1–12, 2000.
72. F. Hoffmeister and T. Bäck. Genetic algorithms and evolution strategies: Similarities and differences. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1991.
73. J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

74. J.N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42, 1995.
75. H.H. Hoos and T. Stützle. *Stochastic Local Search – Foundations and Applications*. Elsevier, Amsterdam, 2005.
76. T. Ibaraki, K. Nonobe, and M. Yagiura, editors. *Metaheuristics: Progress as Real Problem Solvers*. Springer, New York, 2005.
77. L. Ingber. Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics*, 25:33–54, 1996.
78. A. Jaszkiewicz. A comparative study of multiple-objective metaheuristics on the bi-objective set covering problem and the pareto memetic algorithm. *Annals of Operations Research*, 131:215–235, 2004.
79. D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning. *Operations Research*, 37:865–892, 1989.
80. S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
81. M. Laguna and R. Martí. *Scatter Search*. Kluwer, Boston, 2003.
82. S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21:498–516, 1973.
83. C. McGeoch. Toward an experimental method for algorithm simulation. *INFORMS Journal on Computing*, 8:1–15, 1996.
84. C. Meloni, D. Pacciarelli, and M. Pranzo. A rollout metaheuristic for job shop scheduling problems. *Annals of Operations Research*, 131:215–235, 2004.
85. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 3 edition, 1999.
86. Z. Michalewicz and D.B. Fogel. *How to Solve It: Modern Heuristics*. Springer, Berlin, 2 edition, 2004.
87. P. Moscato. An introduction to population approaches for optimization and hierarchical objective functions: A discussion on the role of tabu search. *Annals of Operations Research*, 41:85–121, 1993.
88. I.H. Osman and J.P. Kelly, editors. *Meta-Heuristics: Theory and Applications*. Kluwer, Boston, 1996.
89. M.W. Park and Y.D. Kim. A systematic procedure for setting parameters in simulated annealing algorithms. *Computers & Operations Research*, 25(3):207–217, 1998.
90. R. Parson and M.E. Johnson. A case study in experimental design applied to genetic algorithms with applications to DNA sequence assembly. *American Journal of Mathematical and Management Sciences*, 17(3):369–396, 1997.
91. J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, 1984.
92. E. Pesch and F. Glover. TSP ejection chains. *Discrete Applied Mathematics*, 76:165–182, 1997.
93. G. Polya. *How to solve it*. Princeton University Press, Princeton, 1945.
94. J. Puchinger and G.R. Raidl. An evolutionary algorithm for column generation in integer programming: an effective approach for 2D bin packing. In X. Yao, E.K. Burke, J.A. Lozano, J. Smith, J.J. Merelo-Guervos, J.A. Bullinaria, J.E. Rowe, P. Tino, A. Kaban, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 642–651. Springer Verlag, 2004.
95. J. Puchinger and G.R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In J. Mira and J.R. Álvarez, editors, *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, Part II*, volume 3562 of *Lecture Notes in Computer Science*, pages 41–53. Springer, 2005.

96. G.R. Raidl. A unified view on hybrid metaheuristics. In F. Almeida, M.J. Blesa, C. Blum, J.M. Moreno-Vega, M.M. Pérez, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2006.
97. B. Rangaswamy, A. S. Jain, and F. Glover. Tabu search candidate list strategies in scheduling. pages 215–233, 1998.
98. V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, editors. *Modern Heuristic Search Methods*. Wiley, Chichester, 1996.
99. C.R. Reeves and J.E. Rowe. *Genetic Algorithms: Principles and Perspectives*. Kluwer, Boston, 2002.
100. C. Rego and B. Alidaee, editors. *Metaheuristic Optimization via Memory and Evolution*. 2005.
101. M.G.C. Resende and J.P. de Sousa, editors. *Metaheuristics: Computer Decision-Making*. Kluwer, Boston, 2004.
102. C.C. Ribeiro and P. Hansen, editors. *Essays and Surveys in Metaheuristics*. Kluwer, Boston, 2002.
103. F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier, 2006.
104. R.Y. Rubinstein. Optimization of Computer Simulation Models with Rare Events. *European Journal of Operational Research*, 99:89–112, 1997.
105. M. Sakawa. *Genetic Algorithms and Fuzzy Multiobjective Optimization*. Kluwer, Boston, 2001.
106. H.-P. Schwefel and T. Bäck. Artificial evolution: How and why? In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science: Recent Advances and Industrial Applications*, pages 1–19. Wiley, Chichester, 1998.
107. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. Working paper, ILOG S.A., Gentilly, France, 1998.
108. K. Smith. Neural networks for combinatorial optimisation: A review of more than a decade of research. *INFORMS Journal on Computing*, 11:15–34, 1999.
109. M. Sniedovich and S. Voß. The corridor method: A dynamic programming inspired metaheuristic. *Control and Cybernetics*, 35:551–578, 2006.
110. L. Sondergeld. *Performance Analysis Methods for Heuristic Search Optimization with an Application to Cooperative Agent Algorithms*. Shaker, Aachen, 2001.
111. R.H. Storer, S.D. Wu, and R. Vaccari. Problem and heuristic space search strategies for job shop scheduling. *ORSA Journal on Computing*, 7:453–467, 1995.
112. E. Taillard and S. Voß. POPMUSIC — partial optimization metaheuristic under special intensification conditions. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 613–629. Kluwer, Boston, 2002.
113. E. Taillard, P. Waelti, and J. Zuber. Few statistical tests for proportions comparison. *European Journal of Operational Research*, 185(3):1336–1350, 2006.
114. É.D. Taillard, L.M. Gambardella, M. Gendreau, and J.-Y. Potvin. Adaptive memory programming: A unified view of meta-heuristics. *European Journal of Operational Research*, 135:1–16, 2001.
115. J. Tavares, F. Pereira, and E. Costa. Multidimensional knapsack problem: A fitness landscape analysis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cynernetics*, 38(3):604–616, 2008.
116. R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra. A local search template. *Computers & Operations Research*, 25:969–979, 1998.
117. M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search techniques. *Journal of Heuristics*, 1:43–65, 1995.
118. S. Voß. *Intelligent Search*. Manuscript, TU Darmstadt, 1993.
119. S. Voß. Tabu search: applications and prospects. In D.-Z. Du and P. Pardalos, editors, *Network Optimization Problems*, pages 333–353. World Scientific, Singapore, 1993.

120. S. Voß. Observing logical interdependencies in tabu search: Methods and results. In V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, editors, *Modern Heuristic Search Methods*, pages 41–59, Chichester, 1996. Wiley.
121. S. Voß. Meta-heuristics: The state of the art. In A. Nareyek, editor, *Local Search for Planning and Scheduling*, volume 2148 of *Lecture Notes in Artificial Intelligence*, pages 1–23. Springer, 2001.
122. S. Voß. Metaheuristics. In C.A. Floudas and P.M. Pardalos, editors, *Encyclopedia of Optimization*. Springer, New York, 2008.
123. S. Voß, A. Fink, and C. Duin. Looking ahead with the pilot method. *Annals of Operations Research*, 136:285–302, 2005.
124. S. Voß, S. Martello, I.H Osman, and C. Roucairol, editors. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, Boston, 1999.
125. S. Voß and D.L. Woodruff, editors. *Optimization Software Class Libraries*. Kluwer, Boston, 2002.
126. J. P. Watson, L. D. Whitley, and A. E. Howe. Linking search space structure, run-time dynamics, and problem difficulty: A step toward demystifying tabu search. *Journal of Artificial Intelligence Research*, 24:221–261, 2005.
127. D. Whitley, S. Rana, J. Dzubera, and K.E. Mathias. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85:245–276, 1996.
128. D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.
129. D.L. Woodruff. Proposals for chunking and tabu search. *European Journal of Operational Research*, 106:585–598, 1998.
130. D.L. Woodruff. A chunking based selection strategy for integrating meta-heuristics with branch and bound. In S. Voß, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 499–511. Kluwer, Boston, 1999.
131. J. Xu, S.Y. Chiu, and F. Glover. Fine-tuning a tabu search algorithm with statistical tests. *International Transactions in Operational Research*, 5(3):233–244, 1998.
132. J.H. Zar. *Biostatistical Analysis*. Prentice Hall, Upper Saddle River, New Jersey, 1999.
133. M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization. *Annals of Operations Research*, 131(1):373–395, 2004.

Chapter 2
Just MIP it!

Matteo Fischetti, Andrea Lodi, and Domenico Salvagnin

Abstract Modern Mixed-Integer Programming (MIP) solvers exploit a rich arsenal of tools to attack hard problems. It is widely accepted by the OR community that the solution of very hard MIPs can take advantage from the solution of a series of time-consuming auxiliary Linear Programs (LPs) intended to enhance the performance of the overall MIP solver. For instance, auxiliary LPs may be solved to generate powerful disjunctive cuts, or to implement a strong branching policy. Also well established is the fact that finding good-quality heuristic MIP solutions often requires a computing time that is just comparable to that needed to solve the LP relaxations. So, it makes sense to think of a new generation of MIP solvers where auxiliary MIPs (as opposed to LPs) are heuristically solved on the fly, with the aim of bringing the MIP technology under the chest of the MIP solver itself. This leads to the idea of "translating into a MIP model" (*MIPping*) some crucial decisions to be taken within a MIP algorithm (How to cut? How to improve the incumbent solution? Is the current node dominated?). In this paper we survey a number of successful applications of the above approach.

Matteo Fischetti
DEI, Università di Padova, Padua, Italy
e-mail: matteo.fischetti@unipd.it

Andrea Lodi
DEIS, Università di Bologna, Bologna, Italy
e-mail: andrea.lodi@unibo.it

Domenico Salvagnin
DMPA, Università di Padova, Padua, Italy
e-mail: dominiqs@gmail.com

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_2,

2.1 Introduction

Modern MIP solvers exploit a rich arsenal of tools to attack hard problems. Some successful examples involve the solution of LP models to control the branching strategy (strong branching), the cut generation (lift-and-project), and the heuristics (reduced costs). As a matter of fact, it is well known by the OR community that the solution of very hard MIPs can take advantage of the solution of a series of auxiliary LPs intended to guide the main steps of the MIP solver.

Also well known is the fact that finding good-quality heuristic MIP solutions often requires a computing time that is just comparable to that needed to solve the LP relaxation of the problem at hand. This leads to the idea of "translating into a MIP model" (*MIPping*) some crucial decisions to be taken within a MIP algorithm (in particular: How to cut? How to improve the incumbent solution? Is the current node dominated?), with the aim of bringing the MIP technology well inside the MIP solver.

The present paper gives a survey of three successful applications of the MIPping approach. In Section 2.2 we address the generation of strong cutting planes. In this context, the MIPping approach has been extensively applied to modeling and solving (possibly in a heuristic way) the NP-hard separation problems of famous classes of valid inequalities for mixed integer linear programs. Besides the theoretical interest in evaluating the strength of these classes of cuts computationally, the approach proved successful also in practice, and allowed the solution of very hard MIPLIB instances [2] that could not be solved before.

In Section 2.3 we address enhanced (primal) heuristic approaches for the solution of hard MIP models. An example of the benefits deriving from the use of a black-box MIP solver to produce heuristic primal solutions for a generic MIP is the recently-proposed *local branching* paradigm that uses a general-purpose MIP solver to explore large solution neighborhoods defined through the introduction in the MIP model of invalid linear inequalities called *local branching cuts* [25]. More recently, a different heuristic approach called *Feasibility Pump* has been proposed to address the problem of finding an initial feasible solution and of improving it. In Section 2.3 we describe a hybrid algorithm that uses the feasibility pump method to provide, at very low computational cost, an initial (possibly infeasible) solution to the local branching procedure.

In Section 2.4 we finally address the general-purpose dominance procedure proposed in the late 80's by Fischetti and Toth [30], that overcomes some of the drawbacks of the classical dominance definition. Given the current node α of the search tree, let J^α be the set of variables fixed to some value. Following the MIPping paradigm, we construct an auxiliary problem XP^α that looks for a new partial assignment involving the variables in J^α and such that (i) the objective function value is not worse than the one associated with the original assignment, and (ii) every completion of the old partial assignment

is also a valid completion of the new one. If such a new partial assignment is found (and a certain tie-breaking rule is satisfied), one is allowed to fathom node α.

The present survey is based on previous published work; in particular, Sections 2.2, 2.3 and 2.4 are largely based on [26], [28] and [54], respectively.

2.2 MIPping Cut Separation

In this section we first introduce our basic notation and definitions and review some classical results on cutting planes for pure and mixed integer problems. Then, we discuss in Section 2.2.1 the separation of pure integer cuts, i.e., those cuts in which (i) all coefficients are integer and (ii) continuous variables (if any) have null coefficients. In Section 2.2.2 we address the more general (and powerful) family of split cuts which are instead mixed integer inequalities because the two conditions above do not apply. Finally, in Subsection 2.2.3 we discuss computational aspects of these models and we report results on the strength of the addressed cuts.

Consider first the pure integer linear programming problem $\min\{c^T x : Ax \leq b, x \geq 0, x \text{ integral}\}$ where A is an $m \times n$ rational matrix, $b \in \mathbb{Q}^m$, and $c \in \mathbb{Q}^n$, along with the two associated polyhedra $P := \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ and $P_I := conv\{x \in \mathbb{Z}^n_+ : Ax \leq b\} = conv(P \cap \mathbb{Z}^n)$.

A *Chvátal-Gomory* (*CG*) *cut* (also known as *Gomory fractional* cut) [35, 13] is an inequality of the form $\lfloor u^T A \rfloor x \leq \lfloor u^T b \rfloor$ where $u \in \mathbb{R}^m_+$ is a vector of multipliers, and $\lfloor \cdot \rfloor$ denotes the lower integer part. Chvátal-Gomory cuts are valid inequalities for P_I. The *Chvátal closure* of P is defined as

$$P^1 := \{x \geq 0 : Ax \leq b, \lfloor u^T A \rfloor x \leq \lfloor u^T b \rfloor \text{ for all } u \in \mathbb{R}^m_+\}. \tag{2.1}$$

Thus $P_I \subseteq P^1 \subseteq P$. By the well-known equivalence between optimization and separation [37], optimizing over the first Chvátal closure is equivalent to solving the *CG separation problem* where we are given a point $x^* \in \mathbb{R}^n$ and are asked to find a hyperplane separating x^* from P^1 (if any). Without loss of generality we can assume that $x^* \in P$, since all other points can be cut by simply enumerating the members of the original inequality system $Ax \leq b$, $x \geq 0$. Therefore, the separation problem we are actually interested in reads:

> **CG-SEP**: Given any point $x^* \in P$ find (if any) a CG cut that is violated by x^*, i.e., find $u \in \mathbb{R}^m_+$ such that $\lfloor u^T A \rfloor x^* > \lfloor u^T b \rfloor$, or prove that no such u exists.

It was proved by Eisenbrand [23] that CG-SEP is NP-hard, so optimizing over P^1 also is.

Moreover, Gomory [36] proposed a stronger family of cuts, the so-called *Gomory Mixed Integer (GMI) cuts*, that apply to both the pure integer and

the mixed integer case. Such a family of inequalities has been proved to be equivalent to two other families, the so-called *split cuts* defined by Cook et al. [15], and the *Mixed Integer Rounding (MIR) cuts* introduced by Nemhauser and Wolsey [50]. The reader is referred to Cornuéjols and Li [17] for formal proofs of the correspondence among those families, and to Cornuéjols [16] for a very recent survey on valid inequalities for mixed integer linear programs. Let us consider a generic MIP of the form:

$$\min\{c^T x + f^T y : Ax + Cy \leq b, x \geq 0, x \text{ integral}, \ y \geq 0\} \tag{2.2}$$

where A and C are $m \times n$ and $m \times r$ rational matrices respectively, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, and $f \in \mathbb{Q}^r$. We also consider the two following polyhedra in the (x, y)-space:

$$P(x, y) := \{(x, y) \in \mathbb{R}^n_+ \times \mathbb{R}^r_+ : Ax + Cy \leq b\}, \tag{2.3}$$

$$P_I(x, y) := conv(\{(x, y) \in P(x, y) : x \text{ integral}\}). \tag{2.4}$$

Split cuts are obtained as follows. For any $\pi \in \mathbb{Z}^n$ and $\pi_0 \in \mathbb{Z}$, the disjunction $\pi^T x \leq \pi_0$ or $\pi^T x \geq \pi_0 + 1$ is of course valid for $P_I(x, y)$, i.e., $P_I(x, y) \subseteq conv(\Pi_0 \cup \Pi_1)$ where

$$\Pi_0 := P(x, y) \cap \{(x, y) : \pi^T x \leq \pi_0\}, \tag{2.5}$$

$$\Pi_1 := P(x, y) \cap \{(x, y) : \pi^T x \geq \pi_0 + 1\}. \tag{2.6}$$

A valid inequality for $conv(\Pi_0 \cup \Pi_1)$ is called a *split cut*. The convex set obtained by intersecting $P(x, y)$ with all the split cuts is called the *split closure* of $P(x, y)$. Cook et al. proved that the split closure of $P(x, y)$ is a polyhedron.

Nemhauser and Wolsey [50] introduced the family of *MIR cuts*, whose basic (2-dimensional) version can be obtained in the following way. Let $1 < \hat{b} < 0$ and $\bar{b} \in Z$, and consider the two-variable mixed integer program $T = \{(x, y) : x + y \geq \hat{b} + \bar{b}, y \geq 0\}$. Then, it is easily seen that the points in T with $x \in Z$ satisfy the *basic MIR* inequality:

$$\hat{b}x + y \geq \hat{b}(\bar{b} + 1), \tag{2.7}$$

that turns out to be a split cut derived from the disjunction $x \leq \bar{b}$ and $x \geq \bar{b}+1$. The hardness of separation of split cuts (and hence of MIR inequalities) has been established by Caprara and Letchford [11].

While Chvátal-Gomory cuts are by definition integer inequalities, split/ GMI/MIR inequalities are instead mixed integer cuts in the sense that the coefficients are generally not integer and the continuous variables (if any) might have nonzero coefficients.

2.2.1 Pure Integer Cuts

As just mentioned, Chvátal-Gomory cuts are of course pure integer inequalities because of the rounding mechanism and since they do apply only to the pure integer case. In the following section we discuss their separation through a MIP model while in Section 2.2.1.2 we show that a closely related model has been used to separate a new class of pure integer cuts for mixed integer problems.

2.2.1.1 Chvátal-Gomory Cuts

Fischetti and Lodi [27] addressed the issue of evaluating the practical strength of P^1 in approximating P_I. The approach was to model the CG separation problem as a MIP, which is then solved through a general-purpose MIP solver. To be more specific, given an input point $x^* \in P$ to be separated[1], CG-SEP calls for a CG cut $\alpha^T x \leq \alpha_0$ which is (maximally) violated by x^*, where $\alpha = \lfloor u^T A \rfloor$ and $\alpha_0 = \lfloor u^T b \rfloor$ for some $u \in \mathbb{R}^m_+$. Hence, if A_j denotes the jth column of A, CG-SEP can be modeled as:

$$\begin{aligned}
\max \quad & \alpha^T x^* - \alpha_0 && (2.8)\\
& \alpha_j \leq u^T A_j \quad \forall j = 1, \ldots, n && (2.9)\\
& \alpha_0 + 1 - \epsilon \geq u^T b && (2.10)\\
& u_i \geq 0 \qquad \forall i = 1, \ldots, m && (2.11)\\
& \alpha_j \quad \text{integer} \quad \forall j = 0, \ldots, n, && (2.12)
\end{aligned}$$

where ϵ is a small positive value. In the model above, the integer variables α_j $(j = 1, \ldots, n)$ and α_0 play the role of coefficients $\lfloor u^T A_j \rfloor$ and $\lfloor u^T b \rfloor$ in the CG cut, respectively. Hence the objective function (2.8) gives the amount of violation of the CG cut evaluated for $x = x^*$, that is what has to be maximized. Because of the sign of the objective function coefficients, the rounding conditions $\alpha_j = \lfloor u^T A_j \rfloor$ can be imposed through upper bound conditions on variables α_j $(j = 1, \ldots, n)$, as in (2.9), and with a lower bound condition on α_0, as in (2.10). Note that this latter constraint requires the introduction of a small value ϵ so as to prevent an integer $u^T b$ being rounded to $u^T b - 1$.

Model (2.8)-(2.12) can also be explained by observing that $\alpha^T x \leq \alpha_0$ is a CG cut if and only if (α, α_0) is an integral vector, as stated in (2.12), and $\alpha^T x \leq \alpha_0 + 1 - \epsilon$ is a valid inequality for P, as stated in (2.9)-(2.11) by using the well-known characterization of valid inequalities for a polyhedron due to Farkas.

[1] Recall that Gomory's work [35] implies that CG-SEP is easy when x^* is an extreme point of P.

2.2.1.2 Projected Chvátal-Gomory Cuts

Bonami et al. [10] extended the concept of Chvátal-Gomory cuts to the mixed integer case. Such an extension is interesting in itself and has the advantage of identifying a large class of cutting planes whose resulting separation problem retains the simple structure of model (2.8)-(2.12) above. One can define the projection of $P(x, y)$ onto the space of the x variables as:

$$P(x) := \{x \in \mathbb{R}^n_+ : \text{there exists } y \in \mathbb{R}^r_+ \text{ s.t. } Ax + Cy \leq b\} \quad (2.13)$$

$$= \{x \in \mathbb{R}^n_+ : u^k A \leq u^k b,\ k = 1, \ldots, K\} \quad (2.14)$$

$$=: \{x \in \mathbb{R}^n_+ : \bar{A}x \leq \bar{b}\}, \quad (2.15)$$

where $u^1, \ldots, u^K$ are the (finitely many) extreme rays of the projection cone $\{u \in \mathbb{R}^m_+ : u^T C \geq 0^T\}$. Note that the rows of the linear system $\bar{A}x \leq \bar{b}$ are of Chvátal rank 0 with respect to $P(x, y)$, i.e., no rounding argument is needed to prove their validity.

We define a *projected Chvátal-Gomory (pro-CG) cut* as a CG cut derived from the system $\bar{A}x \leq \bar{b},\ x \geq 0$, i.e., an inequality of the form $\lfloor w^T \bar{A} \rfloor x \leq \lfloor w^T \bar{b} \rfloor$ for some $w \geq 0$. Since any row of $\bar{A}x \leq \bar{b}$ can be obtained as a linear combination of the rows of $Ax \leq b$ with multipliers $\bar{u} \geq 0$ such that $\bar{u}^T C \geq 0^T$, it follows that a pro-CG cut can equivalently (and more directly) be defined as an inequality of the form:

$$\lfloor u^T A \rfloor x \leq \lfloor u^T b \rfloor \quad \text{for any } u \geq 0 \text{ such that } u^T C \geq 0^T. \quad (2.16)$$

As such, its associated separation problem can be modeled as a simple extension of the system (2.8)-(2.12) by amending it through the following set of inequalities:

$$u^T C_j \geq 0 \quad \forall j = 1, \ldots, r. \quad (2.17)$$

Projected Chvátal-Gomory cuts are dominated by split cuts, and therefore $P^1(x, y)$ contains the split closure of $P(x, y)$. More precisely, $P^1(x, y)$ is the intersection of $P(x, y)$ with all the split cuts where one of the sets Π_0, Π_1 defined in (2.5) and (2.6) is empty (see [10]).

2.2.2 Mixed Integer Cuts

The computational results reported in [27] and [10] showed that P^1 often gives a surprisingly tight approximation of P_I, thus triggering research in the attempt of extending the approach to (more powerful) mixed integer cuts.

Unfortunately, model (2.8)-(2.12) does not extend immediately to the mixed integer case if one wants to concentrate on split/MIR/GMI cuts where coefficients are not necessarily integer and the continuous variables might as-

sume nonzero coefficients in the cut. A natural mixed integer nonlinear model has been suggested in [11]. Variants of such a model have been solved with two different approaches: by solving either a parametric mixed integer problem [7] (Section 2.2.2.1) or a nonlinear mixed integer problem [19, 20] (Section 2.2.2.2).

Finally, it is not difficult to see that one can use the multipliers u computed as in (2.8)-(2.12) or (2.8)-(2.12),(2.17) and write a GMI inequality instead of a CG or pro-CG cut. However, such an *a posteriori* strengthening did not turn out to be very effective (see [10]).

2.2.2.1 Split Cuts Solving a Parametric MIP

Balas and Saxena [7] directly addressed the separation problem of the most violated split cut of the form $\alpha^T x + \gamma^T y \geq \beta$ by looking at the union of the two polyhedra (2.5) and (2.6). In particular, they addressed a generic MIP of the form:

$$\min\{c^T x + f^T y : Ax + Cy \geq b, x \text{ integral}\}, \tag{2.18}$$

where the variable bounds are included among the explicit constraints, and wrote a first nonlinear separation model for split cuts as follows:

$$\begin{aligned}
\min \quad & \alpha^T x^* + \gamma^T y^* - \beta && && (2.19)\\
& \alpha_j = u^T A_j - u_0 \pi_j && \forall j = 1, \ldots, n && (2.20)\\
& \gamma_j = u^T C_j && \forall j = 1, \ldots, r && (2.21)\\
& \alpha_j = v^T A_j + v_0 \pi_j && \forall j = 1, \ldots, n && (2.22)\\
& \gamma_j = v^T C_j && \forall j = 1, \ldots, r && (2.23)\\
& \beta = u^T b - u_0 \pi_0 && && (2.24)\\
& \beta = v^T b + v_0(\pi_0 + 1) && && (2.25)\\
& 1 = u_0 + v_0 && && (2.26)\\
u, \ v, \ u_0, \ v_0 & \geq 0 && && (2.27)\\
\pi, \ \pi_0 \quad & \text{integer.} && && (2.28)
\end{aligned}$$

Normalization constraint (2.26) allows one to simplify the model to the form below:

$$\begin{aligned}
\min & \, u^T(Ax^* + Cy^* - b) - u_0(\pi^T x^* - \pi_0) && && (2.29)\\
& u^T A_j - v^T A_j - \pi_j = 0 && \forall j = 1, \ldots, n && (2.30)\\
& u^T C_j - v^T C_j = 0 && \forall j = 1, \ldots, r && (2.31)\\
& -u^T b + v^T b + \pi_0 = u_0 - 1 && && (2.32)\\
& 0 < u_0 < 1 \ , \ u, v \geq 0 && && (2.33)\\
& \pi, \ \pi_0 \quad \text{integer}, && && (2.34)
\end{aligned}$$

where v_0 has been removed by using constraint (2.26), and one explicitly uses the fact that any nontrivial cut has $u_0 < 1$ and $v_0 < 1$ (see Balas and Perregaard [6]). Note that the nonlinearity only arises in the objective function. Moreover, for any fixed value of parameter u_0 the model becomes a regular MIP.

The continuous relaxation of the above model yields a parametric linear program which can be solved by a variant of the simplex algorithm (see, e.g., Nazareth [49]). Balas and Saxena [7] however avoided solving the parametric MIP through a specialized algorithm, and considered a grid of possible values for parameter u_0, say $u_0^1 < u_0^2 < \cdots < u_0^k$. The grid is initialized by means of the set $\{0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$ and then is enriched, on the fly, by bisecting a certain interval $[u_0^t, u_0^{t+1}]$ through the insertion of the new grid point $u_0' := (u_0^t + u_0^{t+1})/2$.

2.2.2.2 Split Cuts Solving a Nonlinear MIP

Dash et al. [19, 20] addressed the optimization over the split closure by looking at the corresponding MIR inequalities and, more precisely, developed a mixed integer nonlinear model and linearized it in an effective way.

For the ease of writing the model, we slightly change the definition of polyhedron $P(x, y)$ by putting the constraints in equality form as:

$$P(x,y) = \{(x,y) \in \mathbb{R}^n_+ \times \mathbb{R}^r_+ \ : \ Ax + Cy + Is = b, \ s \geq 0\}, \tag{2.35}$$

through the addition of nonnegative slack variables s.

One is looking for an MIR inequality in the form:

$$u^+ s + \hat{\gamma}^T y + (\hat{\alpha}^T + \hat{\beta}\bar{\alpha}^T)x \geq \hat{\beta}(\bar{\beta} + 1), \tag{2.36}$$

where $\bar{\alpha}$ and $\bar{\beta}$ are vectors of integer variables, u^+, $\hat{\alpha}$ and $\hat{\gamma}$ are vectors of nonnegative variables, and $0 < \hat{\beta} < 1$.

Let $\sum_{k \in K} \epsilon_k < 1$ (e.g., $\epsilon_k = 2^{-k}$). We approximate $\hat{\beta}$ with $\sum_{k \in \bar{K}} \epsilon_k$ for some $\bar{K} \subset K$ and write the RHS of the MIR inequality as $\sum_{k \in \bar{K}} \epsilon_k \Delta$ where $\Delta = \bar{\beta} + 1 - \bar{\alpha}^T x^*$. Using the fact that there is a violated MIR inequality if and only if there is one with $\Delta < 1$, we have the following formulation for the separation of the most violated MIR inequality, where for each $k \in K$ we set $\pi_k = 1$ if $k \in \bar{K}$, and $\pi_k = 0$ otherwise.

$$\min u^+ s^* - \epsilon^T \Phi + \hat{\gamma}^T y^* + \hat{\alpha}^T x^* \tag{2.37}$$
$$\hat{\gamma}_j \geq u^T C_j \quad \forall j = 1, \ldots, r \tag{2.38}$$
$$\hat{\alpha}_j + \bar{\alpha}_j \geq u^T A_j \quad \forall j = 1, \ldots, n \tag{2.39}$$
$$\hat{\beta} + \bar{\beta} \leq u^T b \tag{2.40}$$
$$\hat{\beta} = \sum_{k \in K} \epsilon_k \pi_k \tag{2.41}$$
$$\Delta = (\bar{\beta} + 1) - \bar{\alpha}^T x^* \tag{2.42}$$
$$\Phi_k \leq \Delta \qquad \forall k \in K \tag{2.43}$$
$$\Phi_k \leq \pi_k \qquad \forall k \in K \tag{2.44}$$
$$u_i^+ \geq u_i \qquad \forall i = 1, \ldots, M \tag{2.45}$$
$$u^+,\ \hat{\alpha},\ \hat{\beta}, \hat{\gamma} \geq 0 \tag{2.46}$$
$$\bar{\alpha},\ \bar{\beta} \text{ integer}, \quad \pi \in \{0,1\}^{|K|} \tag{2.47}$$

where $u_i^+ = \max\{u_i, 0\}$ and $M := \{i : s_i^* > 0, i = 1, \ldots, m\}$, i.e., we define a variable u_i^+ only if the corresponding constraint i written in 'less or equal form' is not tight.

The validity of inequality (2.36) can be easily shown. Inequality $u^T s + (\bar{\alpha}^T + \hat{\alpha}^T)x + \hat{\gamma}^T y \geq u^T b$ is valid by Farkas derivation. It remains of course valid by replacing $u_i, \forall i \in M$ with u_i^+ and then one can use the basic MIR inequality (2.7) to obtain the MIR form (2.36) by having as a continuous (nonnegative) part the term $u^+ s + \hat{\alpha}^T x + \hat{\gamma}^T y$.

The approximate model (2.37)–(2.47) turns out to be an exact model if K is chosen appropriately (see [19, 20]).

2.2.3 A Computational Overview

In this section we discuss some simple issues that turn out to be crucial to make the presented models solvable and the MIPping approach successful. Moreover, we show their strength by reporting computational results on MIPs included in the MIPLIB 3.0 [9].

2.2.3.1 Making the Models Solvable

All papers discussed in the previous sections implement pure cutting plane approaches in which (as usual) the following steps are iteratively repeated:

1. the continuous relaxation of the mixed integer program at hand is solved;
2. the separation problem is (heuristically) solved and a set of violated constraints is eventually found;
3. the constraints are added to the original formulation.

Of course, the original formulation becomes larger and larger but in order to provide cuts of rank 1, the separation problem solved at step 2 above only uses the original constraints in the cut derivation. For what concerns the solution of those separation problems, it is important that state-of-the-art MIP solvers such as `ILOG-Cplex` or `Xpress Optimizer` are used, as they incorporate very powerful heuristics that are able to find (and then improve) feasible solutions in short computing times. Indeed, good heuristic solutions are enough for step 2 above, where the NP-hard separation problem does not need to be solved to optimality[2] since any feasible solution provides a valid inequality cutting off the current solution of step 1 above.

In order to make these MIPs solvable, a few issues have to be addressed.

All authors noted that only integer variables in the support of the fractional solution of step 1 above have to be considered, e.g., a constraint $\alpha_j \leq u^T A_j$ for j such that $x^*_j = 0$ is redundant because α_j (times x^*_j) does not contribute to the violation of the cut, while it can be computed a posteriori by an efficient post-processing procedure. It is easy to see that this is also the case of integer variables whose value is at the upper bound, as these variables can be complemented before separation.

The ultimate goal of the cutting plane sketched above is to find, for each fractional point (x^*, y^*) to be cut off, a "round" of cuts that are significantly violated and whose overall effect is as strong as possible in improving the current LP relaxation. A major practical issue for accomplishing such a goal is the strength of the returned cuts. As a matter of fact, several equivalent solutions of the separation problems typically exist, some of which produce very weak cuts for the MIP model. This is because the separation problem actually considers the face $F(x^*, y^*)$ of P_I where all the constraints that are tight at (x^*, y^*) (including the variable bounds) are imposed as equalities. Hence, for this face there exist several formulations of each cut, which are equivalent for $F(x^*, y^*)$ but not for P_I.

The computational experiments in [27] have shown a relation between the strength of a cut and the sparsity of the vector of multipliers u generating it. In particular, the introduction of a penalty term $-\sum_i w_i u_i$ (where i denotes the index of a constraint) in the objective function (2.8), has the effect of making the cut itself sparser. The sparser the cuts the better for the LP problems solved on step 1 of the cutting plane procedure.[3] The importance of making the cuts as sparse as possible has been also documented by Balas and Saxena [7], who noticed that split disjunctions with sparse support tend to give rise to sparse split cuts.

Another important issue in order to accelerate the cutting plane procedure is the cut selection, i.e., finding a set of cuts whose overall behavior is as effective as possible. Cut selection is somehow related to finding a set of cuts which are "as diverse as possible", possibly more effective together. One can

[2] Except eventually in the last step, in which one needs a proof that no additional violated cut exists.

[3] The same sparsification trick is also used in Bonami et al. [10].

Table 2.1 Results for 25 *pure* integer linear programs in the MIPLIB 3.0.

		Split closure	CG closure
% Gap closed	Average	71.71	62.59
% Gap closed	98-100	9 instances	9 instances
% Gap closed	75-98	4 instances	2 instances
% Gap closed	25-75	6 instances	7 instances
% Gap closed	< 25	6 instances	7 instances

expect that such kind of diversification can be strongly improved with cuts obtained by heuristically solving two or more of the discussed separation models; promising results in this direction have been obtained by combining either CG or pro-CG cuts with MIR inequalities [19, 20].

2.2.3.2 Strength of the Closures

The strength of the closures, namely CG, pro-CG and split (or MIR), have been evaluated by running cutting plane algorithms for large (sometimes huge) computing times. Indeed, the goal of the investigation was in all cases to show the tightness of the closures, rather than investigating the practical relevance of the separation MIPping idea when used within a MIP solver. On the other hand, as discussed in the previous section, several techniques can be implemented to speed up the computation and, even in the current status, the MIPping separation approach is not totally impractical. Indeed, one can easily implement a hybrid approach in which the MIP-based separation procedures are applied (for a fixed amount of time) in a preprocessing phase, resulting in a tighter MIP formulation to be solved at a later time by a standard MIP solver. Using this idea, two unsolved MIPLIB 2003 [2] instances, namely `nsrand-ipx` and `arki001`, have been solved to proven optimality for the first time by Fischetti and Lodi [27] and by Balas and Saxena [7], respectively. In other words, for very difficult and challenging problems it does pay to improve the formulation by adding cuts in these closures before switching to either general- or special-purpose solution algorithms.

In Tables 2.1 and 2.2 we report, in an aggregated fashion, the tightness of the closures for MIPLIB 3.0 [9] instances, in terms of percentage of gap closed[4] for pure integer and mixed integer linear programs, respectively.

Most of the results reported in the previous tables give a lower approximation of the exact value of the closures[5], due to the time limits imposed on the cutting plane algorithms. Nevertheless, the picture is pretty clear and

[4] Computed as $100 - 100(opt\_value(P_I) - opt\_value(P^1))/(opt\_value(P_I) - opt\_value(P))$.

[5] In particular, the time limit in [10] to compute a bound of the pro-CG closure is rather short, 20 CPU minutes, and there are pathological instances for which such a closure is ineffective, see [10] for details.

Table 2.2 Results for 33 *mixed* integer linear programs in the MIPLIB 3.0.

		Split closure	pro-CG closure
% Gap closed	Average	84.34	36.38
% Gap closed	98-100	16 instances	3 instances
% Gap closed	75-98	10 instances	3 instances
% Gap closed	25-75	2 instances	11 instances
% Gap closed	< 25	5 instances	17 instances

shows that, although one can construct examples in which the rank of the facets for a polyhedron is very large, in most practical cases the inequalities of rank 1 already give a very tight approximation of the convex hull of integer and mixed integer programs.

2.3 MIPping Heuristics

In this section we consider the problem of finding a feasible (primal) solution to a generic mixed-integer linear program with 0-1 variables of the form:

$$
\begin{aligned}
(P)\quad & \min c^T x && (2.48)\\
\text{s.t.}\quad & Ax \le b && (2.49)\\
& x_j \in \{0,1\}\ \forall j \in \mathcal{B} \neq \emptyset && (2.50)\\
& x_j \ge 0,\ \text{integer}\ \forall j \in \mathcal{G} && (2.51)\\
& x_j \ge 0\ \forall j \in \mathcal{C}, && (2.52)
\end{aligned}
$$

where A is an $m \times n$ input matrix, and b and c are input vectors of dimension m and n, respectively. Here, the variable index set $\mathcal{N} := \{1, \ldots, n\}$ is partitioned into $(\mathcal{B}, \mathcal{G}, \mathcal{C})$, where $\mathcal{B} \neq \emptyset$ is the index set of the 0-1 variables, while the possibly empty sets $\mathcal{G}$ and $\mathcal{C}$ index the general integer and the continuous variables, respectively. Note that we assume the existence of 0-1 variables, as one of the components of the method we actually implemented (namely, the local branching heuristic) is based on this assumption. Our approach can, however, be extended to remove this limitation, as outlined in the concluding remarks of [25]. Also note that constraints (2.49), though stated as inequalities, can involve equalities as well. Let $\mathcal{I} := \mathcal{B} \cup \mathcal{G}$ denote the index set of all integer-constrained variables.

Heuristics for general-purpose MIPs include [4, 5, 8, 18, 22, 29, 32, 33, 34, 38, 39, 40, 43, 44], among others. Recently, we proposed in [25] a heuristic approach, called *Local Branching* (`LB`), to improve the quality of a given feasible solution. This method, as well as other refining heuristics (including the recently-proposed RINS approach [18]), requires the availability of a starting

feasible solution, which is an issue for some difficult MIPs. This topic was investigated by Fischetti et al. [24], who introduced the so-called *Feasibility Pump* (FP) scheme for finding a feasible (or, at least, an "almost feasible") solution to general MIPs through a clever sequence of roundings.

We analyze computationally a simple variant of the original LB method that allows one to deal with infeasible reference solutions, such as those returned by the FP method. Our approach is to start with an "almost feasible" reference solution $\bar{x}$, as available at small computational cost through the FP method. We then relax the MIP model by introducing for each violated constraint: (i) an artificial continuous variable in the constraint itself, (ii) a binary (also artificial) variable, and (iii) a constraint stating that, if the artificial variable has to be used to satisfy the constraint satisfied, then the binary variable must be set to 1. Finally, the objective function is replaced, in the spirit of the first phase of the primal simplex algorithm, by the sum of the artificial binary variables. The initial solution turns out now to be feasible for the relaxed model and its value coincides with the number of initial violated constraints. We then apply the standard LB framework to reduce the value of the objective function, i.e., the number of infeasibilities and a solution of value 0 turns out to be feasible for the initial problem. Note that, although a continuous artificial variable for each violated constraint could be enough, binary variables are better exploited by LB as it will be made clear in Section 2.3.1 and discussed in detail in Section 2.3.2.

Our approach also produces, as a byproduct, a small-cardinality set of constraints whose relaxation (removal) converts a given MIP into a feasible one–a very important piece of information in the analysis of infeasible MIPs. In other words, our method can be viewed as a tool for repairing infeasible MIP *models*, and not just as a heuristic for repairing infeasible MIP *solutions*. This is in the spirit of the widely-studied approaches to find maximum feasible (or minimum infeasible) subsystems of LP models, as addressed e.g. in [3, 12, 31], but is applied here to MIP models. This may be a useful technique in practice.

The section is organized as follows. In Subsection 2.3.1 we review the LB and FP methods. In Subsection 2.3.2 we describe the LB extension we propose to deal with infeasible reference solutions. Computational results are presented in Subsection 2.3.3, where we compare the LB performance with that of the commercial software ILOG-Cplex on two sets of hard 0-1 MIPs, specifically 44 problems taken from the MIPLIB 2003 library [2] and 39 additional instances already considered in [24].

2.3.1 Local Branching and Feasibility Pump

We next review the LB and FP methods. The reader is referred to [25] and [24] for more details.

2.3.1.1 Local Branching

The Local Branching approach works as follows. Suppose a feasible *reference solution* $\bar{x}$ of the MIP is given, and one aims at finding an improved solution that is "not too far" from $\bar{x}$. Let $\overline{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$. For a given positive integer parameter k, we define the *k-OPT neighborhood* $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of the MIP satisfying the additional *local branching constraint*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k, \tag{2.53}$$

where the two terms in the left-hand side count the number of binary variables flipping their value (with respect to $\bar{x}$) either from 1 to 0 or from 0 to 1, respectively. As its name suggests, the local branching constraint (2.53) can be used as a branching criterion within an enumerative scheme for the MIP. Indeed, given the incumbent solution $\bar{x}$, the solution space associated with the current branching node can be partitioned by means of the disjunction:

$$\Delta(x, \bar{x}) \leq k \quad \text{(left branch)} \quad \textbf{or} \quad \Delta(x, \bar{x}) \geq k + 1 \quad \text{(right branch)}, \tag{2.54}$$

where the neighborhood-size parameter k is chosen so as to make the neighborhood $\mathcal{N}(\bar{x}, k)$ "sufficiently small" to be optimized within short computing time, but still "large enough" to likely contain better solutions than $\bar{x}$ (typically, $k = 10$ or $k = 20$).

In [25], we investigated the use of a general-purpose MIP solver as a black-box "tactical" tool to explore effectively suitable solution subspaces defined and controlled at a "strategic" level by a simple external branching framework. The procedure is in the spirit of well-known local search metaheuristics, but the neighborhoods are obtained through the introduction in the MIP model of the local branching constraints (2.53). This allows one to work within a perfectly general MIP framework, and to take advantage of the impressive research and implementation effort that nowadays are devoted to the design of MIP solvers. The new solution strategy is exact in nature, though it is designed to improve the heuristic behavior of the MIP solver at hand. It alternates high-level strategic branchings to define solution neighborhoods, and low-level tactical branchings (performed within the MIP solver) to explore them. The result can then be viewed as a two-level branching strategy aimed at favoring early updatings of the incumbent solution, hence producing improved solutions at early stages of the computation. The computational results reported in [25] show the effectiveness of the `LB` approach. These have also been confirmed by the recent works of Hansen et al. [38] (where `LB` is used within a Variable Neighborhood Search metaheuristic [48]) and of Fischetti et al. [29] (where MIPs with a special structure are investigated).

2.3.1.2 Feasibility Pump

Let $P := \{x \in \mathbb{R}^n : Ax \leq b\}$ denote the polyhedron associated with the LP relaxation of the given MIP, and assume without loss of generality that system $Ax \leq b$ includes the variable bounds:

$$l_j \leq x_j \leq u_j, \quad \forall j \in \mathcal{I},$$

where $l_j = 0$ and $u_j = 1$ for all $j \in \mathcal{B}$. With a little abuse of terminology, we say that a point x is *integer* if $x_j \in Z^n$ for all $j \in \mathcal{I}$ (no matter the value of the other components). Analogously, the rounding $\tilde{x}$ of a given x is obtained by setting $\tilde{x}_j := [x_j]$ if $j \in \mathcal{I}$ and $\tilde{x}_j := x_j$ otherwise, where $[\cdot]$ represents scalar rounding to the nearest integer. The (L_1-norm) distance between a generic point $x \in P$ and a given integer vector $\tilde{x}$ is defined as

$$\Phi(x, \tilde{x}) = \sum_{j \in \mathcal{I}} |x_j - \tilde{x}_j|,$$

(notice that continuous variables $x_j,\ j \notin \mathcal{I}$, if any, are immaterial) and can be modeled as:

$$\Phi(x, \tilde{x}) := \sum_{j \in \mathcal{I}: \tilde{x}_j = l_j} (x_j - l_j) + \sum_{j \in \mathcal{I}: \tilde{x}_j = u_j} (u_j - x_j) + \sum_{j \in \mathcal{I}: l_j < \tilde{x}_j < u_j} (x_j^+ + x_j^-),$$

where the additional variables x_j^+ and x_j^- require the introduction into the MIP model of the additional constraints:

$$x_j = \tilde{x}_j + x_j^+ - x_j^-, \quad x_j^+ \geq 0,\ x_j^- \geq 0, \quad \forall j \in \mathcal{I} : l_j < \tilde{x}_j < u_j. \tag{2.55}$$

It follows that the closest point $x^* \in P$ to $\tilde{x}$ can easily be determined by solving the LP:

$$\min\{\Phi(x, \tilde{x}) : Ax \leq b\}. \tag{2.56}$$

If $\Phi(x^*, \tilde{x}) = 0$, then $x_j^*\ (= \tilde{x}_j)$ is integer for all $j \in \mathcal{I}$, so x^* is a feasible MIP solution. Conversely, given a point $x^* \in P$, the integer point $\tilde{x}$ closest to x^* is easily determined by just rounding x^*.

The `FP` heuristic works with a pair of points $(x^*, \tilde{x})$ with $x^* \in P$ and $\tilde{x}$ integer, that are iteratively updated with the aim of reducing as much as possible their distance $\Phi(x^*, \tilde{x})$. To be more specific, one starts with any $x^* \in P$, and initializes a (typically infeasible) integer $\tilde{x}$ as the rounding of x^*. At each `FP` iteration, called a *pumping cycle*, $\tilde{x}$ is fixed and one finds through linear programming the point $x^* \in P$ which is as close as possible to $\tilde{x}$. If $\Phi(x^*, \tilde{x}) = 0$, then x^* is a MIP feasible solution, and the heuristic stops. Otherwise, $\tilde{x}$ is replaced by the rounding of x^* so as to further reduce $\Phi(x^*, \tilde{x})$, and the process is iterated.

The basic `FP` scheme above tends to stall and stop prematurely. This happens whenever $\Phi(x^*, \tilde{x}) > 0$ is not reduced when replacing $\tilde{x}$ by the rounding

of x^*, meaning that all the integer-constrained components of $\tilde{x}$ remained unchanged in this iteration. In the original FP approach [24], this situation is dealt with by heuristically choosing a few components $\tilde{x}_j$ to be modified, even if this operation increases the current value of $\Phi(x^*, \tilde{x})$. A different approach, to be elaborated in the next section, is to switch to a method based on enumeration, in the attempt to explore a small neighborhood of the current "almost feasible" $\tilde{x}$ (that typically has a very small distance $\Phi(x^*, \tilde{x})$ from P).

2.3.2 LB with Infeasible Reference Solutions

The basic idea of the method presented in this section is that the LB algorithm does not necessarily need to start with a feasible solution—a partially feasible one can be a valid warm start for the method. Indeed, by relaxing the model in a suitable way, it is always possible to consider any infeasible solution, say $\hat{x}$, to be "feasible", and penalize its cost so the LB heuristic can drive it to feasibility.

The most natural way to implement this idea is to add a continuous artificial variable for each constraint violated by $\hat{x}$, and then penalize the use of such variables in the objective function by means of a very large cost M. We tested this approach and found it performs reasonably well on most of the problems. However, it has the drawback that finding a proper value for M may not be easy in practice. Indeed, for a substantial set of problems in the MIPLIB 2003 [2] collection, the value of the objective function is so large that it is difficult to define a value for M that makes any infeasible solution worse than any feasible one. Moreover, the way the LB method works suggests the use of the following, more combinatorial, framework.

Let T be the set of the indices of the constraints $a_i^T x \le b_i$ that are violated by $\hat{x}$. For each $i \in T$, we relax the original constraint $a_i^T x \le b_i$ into $a_i^T x - \sigma_i \le b_i$, where $\sigma_i \ge 0$ is a nonnegative continuous artificial variable, and add the constraint:

$$\sigma_i \le \delta_i y_i, \quad y_i \in \{0, 1\}, \tag{2.57}$$

where δ_i is a sufficiently large value, and y_i is a binary artificial variable attaining value 1 for each violated constraint.[6] Finally, we replace the original objective function $c^T x$ by $\sum_{i \in T} y_i$, so as to count the number of violated constraints. It has to be noted that the set of binary variables in the relaxed model is $\mathcal{B} \cup \mathcal{Y}$, where $\mathcal{Y} := \{y_i : i \in T\}$, hence the structure of the relaxation turns out to be particularly suited for the LB approach, where the

[6] Note that when the violated constraint is in equality form two nonnegative artificial variables, σ_i^+ and σ_i^-, are added with opposite signs and the corresponding constraint (2.57) becomes $\sigma_i^+ + \sigma_i^- \le \delta_i y_i$.

local branching constraint affects precisely the binary variables (including the artificial ones).

An obvious drawback of the method above is that the original objective function is completely disregarded, thus the feasible solution obtained can be arbitrarily bad. A way of avoiding this situation could be to put a term in the artificial objective function that takes the original costs into account. However, a proper balancing of the two contributions (original cost and infeasibility penalty) may not be easy to achieve although promising results in this direction have been reported very recently by Achterberg and Berthold [1]. As a matter of fact, the outcome of a preliminary computational study is that a better overall performance is obtained by using the artificial objective function (alone) until feasibility is reached, and then improving the quality of this solution by using a standard `LB` or RINS approach. This can be done by recovering the original objective function and simply using the computed feasible solution in the usual `LB` way. In other words, the overall algorithm remains in principle exact (see [25] for details) and the proposed scheme is used to provide an initial solution.

2.3.3 Computational Results

In this section, we report on computational results comparing the proposed method with both the `FP` heuristic and the commercial software `ILOG-Cplex` 9.0.3. In our experiments, we used the "asymmetric" version of the local branching constraint (2.53), namely:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j). \tag{2.58}$$

Indeed, as discussed in [25], this version of the constraint seems to be particularly suited for set covering problems where `LB` aims at finding solutions with a small binary support—which is precisely the case of interest in our context.

Our testbed is made up of 33 among the 45 0-1 MIP instances from MIPLIB 2003 [2] and described in Table 2.3, plus an additional set of 39 hard 0-1 MIPs described in Table 2.4. (The 0-1 MIPLIB instance `stp3d` was not considered since the computing time required for the first LP relaxation is larger than one hour, while 11 instances, namely `fixnet6`, `markshare1`, `markshare2`, `mas74`, `mas76`, `modglob`, `pk1`, `pp08a`, `pp08aCUTS`, `set1ch` and `vpm2` have been removed because all tested algorithms found a feasible solution within 0.0 CPU seconds.) The two tables report the name, total number of variables (n), number of 0-1 variables ($|\mathcal{B}|$), and number of constraints (m) for each instance.

Table 2.3 Set of 33 among the 45 0-1 MIP instances collected in MIPLIB 2003 [2].

Name	n	$\|\mathcal{B}\|$	m	Name	n	$\|\mathcal{B}\|$	m
10teams	2025	1800	230	mod011	10958	96	4480
A1C1S1	3648	192	3312	momentum1	5174	2349	42680
aflow30a	842	421	479	net12	14115	1603	14021
aflow40b	2728	1364	1442	nsrand_ipx	6621	6620	735
air04	8904	8904	823	nw04	87482	87482	36
air05	7195	7195	426	opt1217	769	768	64
cap6000	6000	6000	2176	p2756	2756	2756	755
dano3mip	13873	552	3202	protfold	1835	1835	2112
danoint	521	56	664	qiu	840	48	1192
ds	67732	67732	656	rd-rplusc-21	622	457	125899
fast0507	63009	63009	507	seymour	1372	1372	4944
fiber	1298	1254	363	sp97ar	14101	14101	1761
glass4	322	302	396	swath	6805	6724	884
harp2	2993	2993	112	t1717	73885	73885	551
liu	1156	1089	2178	tr12-30	1080	360	750
misc07	260	259	212	van	12481	192	27331
mkc	5325	5323	3411				

Table 2.4 The additional set of 39 0-1 MIP instances.

Name	n	$\|\mathcal{B}\|$	m	source	Name	n	$\|\mathcal{B}\|$	m	source
biella1	7328	6110	1203	[25]	blp-ar98	16021	15806	1128	[43]
NSR8K	38356	32040	6284	[25]	blp-ic97	9845	9753	923	[43]
dc1c	10039	8380	1649	[21]	blp-ic98	13640	13550	717	[43]
dc1l	37297	35638	1653	[21]	blp-ir98	6097	6031	486	[43]
dolom1	11612	9720	1803	[21]	CMS750_4	11697	7196	16381	[42]
siena1	13741	11775	2220	[21]	berlin_5_8_0	1083	794	1532	[42]
trento1	7687	6415	1265	[21]	railway_8_1_0	1796	1177	2527	[42]
rail507	63019	63009	509	[25]	usAbbrv.8.25_70	2312	1681	3291	[42]
rail2536c	15293	15284	2539	[25]	manpower1	10565	10564	25199	[53]
rail2586c	13226	13215	2589	[25]	manpower2	10009	10008	23881	[53]
rail4284c	21714	21705	4284	[25]	manpower3	10009	10008	23915	[53]
rail4872c	24656	24645	4875	[25]	manpower3a	10009	10008	23865	[53]
A2C1S1	3648	192	3312	[25]	manpower4	10009	10008	23914	[53]
B1C1S1	3872	288	3904	[25]	manpower4a	10009	10008	23866	[53]
B2C1S1	3872	288	3904	[25]	ljb2	771	681	1482	[18]
sp97ic	12497	12497	1033	[25]	ljb7	4163	3920	8133	[18]
sp98ar	15085	15085	1435	[25]	ljb9	4721	4460	9231	[18]
sp98ic	10894	10894	825	[25]	ljb10	5496	5196	10742	[18]
bg512142	792	240	1307	[47]	ljb12	4913	4633	9596	[18]
dg012142	2080	640	6310	[47]					

The framework described in the previous section has been tested by using different starting solutions $\hat{x}$ provided by FP. In particular, we wanted to test the sensitivity of our modified LB algorithm with respect to the degree of infeasibility of the starting solution, as well as its capability for improving it. Thus, we executed the FP code for 0, 10 and 100 iterations and passed to LB the integer (infeasible) solution $\hat{x}$ with minimum distance $\Phi(x^*, \hat{x})$ from P. (The case with 0 iterations actually corresponds to starting from the solution

of the continuous relaxation, rounded to the nearest integer.) The resulting three versions of the modified LB are called $\texttt{LB}_0$, $\texttt{LB}_{10}$, and $\texttt{LB}_{100}$, respectively.

In our experiments, we avoided any parameter tuning; FP was implemented exactly as in [24], and for the modified LB code we used a time limit of 30 CPU seconds for the exploration of each local branching neighborhood. As to the value of the neighborhood-size parameter k in LB, we implemented an adaptive procedure: at each neighborhood exploration, we try to reduce the number of violated constraints in the current solution by half, i.e., we set $k = \lfloor |T'|/2 \rfloor$, where $|T'|$ is the value of the current solution. (Since the support of the solution also takes into account non-artificial binary variables, when the number of violated constraints becomes less than 20 we fix $k = 10$, i.e., we use the value suggested in [25] for the asymmetric version of the local branching constraint.) The motivation for this choice is that the number of violated constraints in an initial solution can be extremely large, in which case the use of a small value of k would result in a very slow convergence. A possible drawback is that, in some cases, some of the neighborhoods in the LB sequence can contain no feasible solutions (with respect to the original model) because we do not allow enough artificial variables y to change. The approach can therefore appear counterintuitive, but the idea is that of reducing the neighborhood size iteratively so as to eventually converge.

All codes are written in ANSI C and use the ILOG-Cplex callable libraries. The three modified LB codes ($\texttt{LB}_0$, $\texttt{LB}_{10}$, and $\texttt{LB}_{100}$) are compared with FP and ILOG-Cplex 9.0.3 in Table 2.5 for the MIPLIB 2003 instances, and in Table 2.6 for the additional set of instances. Computing times are expressed in CPU seconds, and refer to a Pentium M 1.6 GHz notebook with 512 MByte of main memory. A time limit of 1,800 CPU seconds was provided for each instance with each algorithm and the computation was halted as soon as a first feasible solution was found.

For each instance, we report in both tables: for ILOG-Cplex, the number of nodes (nodes) needed to find an initial solution and the corresponding computing time (time); for FP, the number of iterations (FPit) and its computing time (time); for each of the three variants of LB, the computing time spent in the FP preprocessing phase (FP time), the initial number of violated constraints ($|T|$), the number of LB iterations (LBit), and the overall computing time (time). Note that we define an LB iteration as the exploration, generally within a time limit, of the neighborhood of the current solution. Moreover, the time reported is the sum of the time of the FP initialization plus the LB time, thus it can be larger than 1,800 CPU seconds. When one of the algorithms was not able to find a feasible solution in the given time limit, we wrote (*) in column "nodes" (for ILOG-Cplex) or "FPit" (for FP), or wrote (μ) in column "$|T|$" near the number of initial infeasible constraints (for LB), where μ is the number of violated constraints in the final solution.

As expected, the degree of infeasibility of the starting solution plays an important role in the LB methods—the better the initial solution, the faster the method. In this view, the FP approach seems to fit particularly well in

Table 2.5 Convergence to a first feasible solution on the MIPLIB 2003 instances.

name	ILOG-Cplex 9.0.3 nodes	time	FP FPit	time	LB_0 FPtime	$\|T\|$	LBit	time	LB_{10} FPtime	$\|T\|$	LBit	time	LB_{100} FPtime	$\|T\|$	LBit	time
10teams	335	**8.4**	70	11.7	0.1	75	29	667.7	1.1	18	11	177.4	11.7	–	–	11.7
A1C1S1	150	4.1	8	3.8	0.1	63	5	**0.8**	3.8	–	–	3.8	3.8	–	–	3.8
aflow30a	0	**0.1**	18	**0.1**	0.0	29	4	3.0	0.1	29	4	0.3	0.1	–	–	**0.1**
aflow40b	370	5.9	6	**0.3**	0.1	40	5	57.6	0.3	–	–	**0.3**	0.3	–	–	**0.3**
air04	40	**8.6**	6	74.7	3.4	125	24	671.8	74.7	–	–	74.7	74.7	–	–	74.7
air05	70	**3.4**	25	83.8	0.8	208	12	135.0	22.8	14	3	25.0	83.8	–	–	83.8
cap6000	0	**0.2**	2	**0.2**	0.1	1	1	**0.2**	0.2	–	–	**0.2**	0.2	–	–	**0.2**
dano3mip	0	**67.7**	2	86.3	65.0	946 (105)	31	1,865.0	86.3	–	–	86.3	86.3	–	–	86.3
danoint	40	1.7	23	**1.5**	0.1	125	6	16.9	0.6	120	5	3.7	1.5	–	–	**1.5**
ds	0	**55.0**	133 (*)	1,800.0	54.5	656	16	582.8	229.9	350	8	302.1	1,358.6	133	6	1,385.0
fast0507	0	**39.0**	3	46.7	43.4	148	1	45.8	46.7	–	–	46.7	46.7	–	–	46.7
fiber	0	0.1	2	**0.0**	0.0	41	5	0.5	0.0	–	–	**0.0**	0.0	–	–	**0.0**
glass4	5389	1.6	124	0.3	0.0	52	5	0.9	0.0	45	4	**0.1**	0.2	45	4	0.3
harp2	0	**0.0**	654	5.0	0.0	9	3	0.9	0.1	6	1	0.1	0.8	6	1	0.8
liu	0	**0.1**	0	**0.1**	0.1	–	–	**0.1**	0.1	–	–	**0.1**	0.1	–	–	**0.1**
misc07	67	**0.2**	78	0.4	0.0	135	7	1.7	0.1	81	6	0.6	0.4	–	–	0.4
mkc	0	**0.2**	2	**0.2**	0.1	9	3	2.2	0.2	–	–	**0.2**	0.2	–	–	**0.2**
mod011	0	0.2	0	**0.1**	0.1	–	–	**0.1**	0.1	–	–	**0.1**	0.1	–	–	**0.1**
momentum1	314 (*)	1,800.0	502	**1,329.6**	1.8	697 (106)	18	1,801.8	42.6	895 (15)	58	1,842.6	178.8	895 (15)	58	1,978.8
net12	203 (*)	1,800.0	1507	225.0	1.8	406	14	246.2	12.9	239	7	**16.8**	21.8	239	7	25.5
nsrand_ipx	0	**0.5**	4	0.9	11.3	390	8	14.1	0.9	–	–	0.9	0.9	–	–	0.9
nw04	0	4.9	1	**4.6**	0.3	6	2	6.8	4.6	–	–	**4.6**	4.6	–	–	**4.6**
opt1217	117	0.1	0	**0.0**	0.0	–	–	**0.0**	0.0	–	–	**0.0**	0.0	–	–	**0.0**
p2756	0	**0.1**	150023 (*)	1,800.0	0.0	41	6	0.8	0.1	19	1	0.2	1.2	19	1	1.3
protfold	190	640.9	367	**502.0**	2.7	37 (37)	7	1,802.7	16.1	13 (1)	50	1,816.1	125.6	7 (1)	55	1,925.6
qiu	0	**0.2**	5	0.3	0.1	132	1	**0.2**	0.3	–	–	0.3	0.3	–	–	0.3
rd-rplusc-21	10978(*)	1,800.0	401 (*)	1,800.0	3.9	119021 (7094)	23	1,803.9	36.8	119017 (1)	71	1,836.8	449.5	119017 (2)	75	2,249.5
seymour	0	**3.5**	7	3.6	3.0	921	1	3.8	3.6	–	–	3.6	3.6	–	–	3.6
sp97ar	0	**3.4**	4	4.2	2.9	222	1	3.8	4.2	–	–	4.2	4.2	–	–	4.2
swath	0	**0.2**	49	2.9	0.1	20	6	124.6	1.0	20	6	70.8	2.9	–	–	2.9
t1717	710	**301.0**	40	814.8	10.7	445 (50)	25	1,810.7	133.2	108 (5)	35	1,933.2	814.8	–	–	814.8
tr12-30	179	0.9	8	**0.1**	0.0	348	8	0.6	0.1	–	–	**0.1**	0.1	–	–	**0.1**
van	0	872.8	10	300.5	27.4	192 (128)	9	1,827.4	**300.5**	–	–	**300.5**	300.5	–	–	**300.5**

Table 2.6 Convergence to a first feasible solution on the additional set of 0-1 MIP instances.

name	ILOG-Cplex 9.0.3 nodes	time	FP FPit	time	LB_0 FPtime	$\|T\|$	LBit	time	LB_{10} FPtime	$\|T\|$	LBit	time	LB_{100} FPtime	$\|T\|$	LBit	time
biella1	594	108.4	4	**2.8**	2.3	1193	9	18.2	2.8	–	–	**2.8**	2.8	–	–	**2.8**
NSR8K	5 (*)	1,800.0	3	**195.5**	185.8	5488 (5488)	1	1,985.8	195.5	–	–	**195.5**	195.5	–	–	**195.5**
dc1c	4749	474.0	2	**12.7**	11.6	1483	11	81.6	12.7	–	–	**12.7**	12.7	–	–	**12.7**
dc1l	0	80.8	2	16.2	14.0	1567	1	**14.8**	16.2	–	–	16.2	16.2	–	–	16.2
dolom1	367	504.4	22	**22.6**	11.9	1410	12	277.1	17.7	632	8	49.4	22.6	–	–	**22.6**
siena1	600	1,371.5	3	**43.7**	40.6	1750	12	271.2	43.7	–	–	**43.7**	43.7	–	–	**43.7**
trento1	340	276.8	7	**11.0**	9.3	603	8	22.6	11.0	–	–	**11.0**	11.0	–	–	**11.0**
rail507	0	32.8	2	8.7	6.5	218	1	**7.4**	8.7	–	–	8.7	8.7	–	–	8.7
rail2536c	0	16.8	1	15.2	14.3	2008	1	**14.9**	15.2	–	–	15.2	15.2	–	–	15.2
rail2586c	0	63.9	1	8.3	7.6	1871	1	**7.9**	8.3	–	–	8.3	8.3	–	–	8.3
rail4284c	0	204.9	2	56.7	53.5	3305	1	**54.2**	56.7	–	–	56.7	56.7	–	–	56.7
rail4872c	0	186.4	2	19.3	17.5	3254	1	**18.3**	19.3	–	–	19.3	19.3	–	–	19.3
A2C1S1	0	**0.1**	5	4.7	0.1	60	1	0.2	4.7	–	–	4.7	4.7	–	–	4.7
B1C1S1	0	**0.1**	6	5.0	0.1	208	1	0.2	5.0	–	–	5.0	5.0	–	–	5.0
B2C1S1	0	**0.1**	7	4.7	0.1	217	1	0.3	4.7	–	–	4.7	4.7	–	–	4.7
sp97ic	0	**2.4**	3	3.1	1.7	173	1	**2.4**	3.1	–	–	3.1	3.1	–	–	3.1
sp98ar	0	**3.8**	3	5.2	3.5	260	7	23.6	5.2	–	–	5.2	5.2	–	–	5.2
sp98ic	0	**2.1**	2	2.6	1.8	147	6	6.0	2.6	–	–	2.6	2.6	–	–	2.6
blp-ar98	8300	158.3	835	122.9	0.5	212	8	31.7	2.5	204	7	**16.4**	15.7	205	7	25.9
blp-ic97	1120	16.2	8	**1.3**	0.3	59	5	5.5	1.3	–	–	**1.3**	1.3	–	–	**1.3**
blp-ic98	1570	33.6	3	**1.5**	0.9	76	5	5.0	1.5	–	–	**1.5**	1.5	–	–	**1.5**
blp-ir98	1230	8.1	4	**0.4**	0.1	37	4	1.3	0.4	–	–	**0.4**	0.4	–	–	**0.4**
CMS750_4	940	27.2	16	**6.5**	0.7	2446	1	9.2	3.3	2441	1	11.7	6.5	–	–	**6.5**
berlin_5_8_0	152	0.4	13	**0.2**	0.0	170	20	275.4	0.1	167	1	**0.2**	0.2	–	–	**0.2**
railway_8_1_0	350	1.3	12	**0.3**	0.1	374	17	358.4	0.2	373	1	0.5	0.3	–	–	**0.3**
usAbbrv.8.25_70	274581	1,371.5	31	0.7	0.1	400	1	**0.6**	0.3	376	1	0.8	0.7	–	–	0.7
bg512142	0	0.3	0	**0.2**	0.2	–	–	**0.2**	0.2	–	–	**0.2**	0.2	–	–	**0.2**
dg012142	0	1.0	0	**0.8**	0.8	–	–	**0.8**	0.8	–	–	**0.8**	0.8	–	–	**0.8**
manpower1	154 (*)	1,800.0	30	**18.8**	8.4	1142	15	108.7	13.4	336	9	52.0	18.8	–	–	**18.8**
manpower2	150	364.6	92	**137.5**	39.5	1181	30	774.2	73.6	309	13	394.8	137.5	–	–	**137.5**
manpower3	181	326.9	42	**76.2**	27.1	1160	23	534.7	55.5	427	17	363.3	76.2	–	–	**76.2**
manpower3a	181	925.1	293	**294.1**	30.6	1327 (7)	57	1,830.6	53.3	369	18	491.2	114.8	9	2	372.1
manpower4	185	671.0	208	**138.9**	14.3	1105	34	1,010.8	41.8	604	19	427.7	80.5	40	8	383.8
manpower4a	194	1,039.9	308	289.2	36.4	1226	37	814.8	69.1	483	18	440.0	159.3	7	4	**206.2**
ljb2	30	0.2	0	**0.0**	0.0	–	–	**0.0**	0.0	–	–	**0.0**	0.0	–	–	**0.0**
ljb7	100	3.8	0	**0.6**	0.6	–	–	**0.6**	0.6	–	–	**0.6**	0.6	–	–	**0.6**
ljb9	180	7.0	0	**0.8**	0.8	–	–	**0.8**	0.8	–	–	**0.8**	0.8	–	–	**0.8**
ljb10	90	5.9	0	**1.1**	1.1	–	–	**1.1**	1.1	–	–	**1.1**	1.1	–	–	**1.1**
ljb12	110	5.8	0	**0.7**	0.7	–	–	**0.7**	0.7	–	–	**0.7**	0.7	–	–	**0.7**

our context, in that it is able to provide very good solutions (as far as the degree of infeasibility is concerned) in very short computing times. Among the three LB implementations, $\texttt{LB}_0$ failed eight times in finding a feasible solution within the time limit, $\texttt{LB}_{10}$ four times, and $\texttt{LB}_0$ only three times. Among the 64 instances for which the three LB implementations found a feasible solution within the time limit, $\texttt{LB}_0$ was at least as fast as the other two in 26 cases, $\texttt{LB}_{10}$ in 34 cases, and $\texttt{LB}_{100}$ in 42 cases. Overall, $\texttt{LB}_{100}$ qualifies as the most effective (and stable) of the three methods.

A comparison between ILOG-Cplex and $\texttt{LB}_{100}$ shows that:

1. ILOG-Cplex was not able to find any feasible solution (within the 1,800 second time limit) in five cases, whereas $\texttt{LB}_{100}$ was unsuccessful three times;
2. among the 66 instances for which both algorithms found a feasible solution within the time limit, ILOG-Cplex was strictly faster in 21 cases, while the opposite holds in 41 cases;
3. among the same 66 instances, the average computing time for finding a feasible solution for ILOG-Cplex was 146.7 CPU seconds, while for $\texttt{LB}_{100}$ it was 65.0 CPU seconds.

As expected, the quality of the initial ILOG-Cplex solution (not reported in the tables) is typically better than that provided by the LB methods. More precisely, the geometric mean of the ratio between the solution found by an algorithm and best solution is 2.28, 2.22, 2.11 and 1.13 for the three LB implementations and ILOG-Cplex, respectively. As noted in Section 2.3.2, however, the first solution can be easily improved by standard use of the LB algorithm. As an example, on instance dc1c the ratio of the solution obtained by algorithm $\texttt{LB}_0$ with respect to the first solution computed by ILOG-Cplex is 12.16. However, the ratio reduces significantly by applying LB, and becomes 4.83, 2.74, and 1.02 in the first three iterations, respectively, and reaches value 0.77 (i.e., the ILOG-Cplex solution is eventually improved) in the fourth one. Those four iterations take 138.1 CPU seconds, plus 81.6 seconds to find the first solution, thus the overall computing time of 219.7 CPU seconds is less than a half of the time spent by ILOG-Cplex to find its first solution, namely 474.0 CPU seconds.

This very satisfactory behavior is unfortunately not confirmed on other instances, though local search heuristics such as LB or RINS [18] generally improve the first solution considerably. On the other hand, an effective exploitation of a first feasible solution within an enumerative algorithm is by itself a relevant (and difficult) research topic, that recently started to receive considerable attention in the field.

2.4 MIPping the Dominance Test

In the standard B&B (or B&C) framework, a node is fathomed in two situations:

1. the LP relaxation of the node is infeasible;
2. the LP relaxation optimum is not better than the value of the incumbent optimal solution.

There is, however, a third way of pruning a node, by using dominances. According to [51], a dominance relation is defined as follows: if we can show that the best descendant of a node β is at least as good as the best descendant of a node α, then we say that node β *dominates* node α, meaning that the latter can be fathomed (in case of ties, an appropriate rule has to be taken into account in order to avoid fathoming cycles). Unfortunately, this definition may become useless in the context of general MIPs, where we do not actually know how to perform the dominance test without storing huge amounts of information for all the previously-generated nodes — which is often impractical.

Fischetti and Toth [30] proposed a different (and more "local") dominance procedure which overcomes many of the drawbacks of the classical definition, and resembles somehow the *isomorphic-pruning* introduced recently by Margot [45]. Here is how the procedure works.

Let the MIP problem at hand denoted as:

$$\min\{c^T x : x \in \mathbb{R}^n_+,\ Ax \le b,\ x_j \text{ integer for all } j \in J\}, \tag{2.59}$$

where $J \subseteq I := \{1, \cdots, n\}$ is the index-set of the integer variables. For any $J' \subseteq J$ and for any $x' \in \mathbb{R}^n_+$, we denote as:

$$c(J', x') := \sum_{j \in J'} c_j x'_j,$$

the contribution of the variables in J' to the overall cost $c^T x'$. Now, let us suppose to solve problem (2.59) by an enumerative (B&B or B&C) algorithm whose branching rule fixes some of the integer-constrained variables to some values. For every node k of the search tree, let $J^k \subseteq J$ denote the set of indices of the variables x_j fixed to a certain value x^k_j (say). Every solution x such that $x_j = x^k_j$ for all $j \in J^k$ (i.e., belonging to the subtree rooted at node k) is called a *completion* of the partial solution associated at node k.

Definition 1. Let α and β be two nodes of the search tree. Node β dominates node α if:

1. $J^\beta = J^\alpha$;
2. $c(J^\beta, x^\beta) \le c(J^\alpha, x^\alpha)$, i.e., the cost of the partial solution at node β is not worse than that at node α;

3. every completion of the partial solution associated with node α is also a completion of the partial solution associated with node β.

Clearly, according to the classical dominance theory, the existence of a node β unfathomed that dominates node α is a sufficient condition to fathom node α. A key question at this point is: Given the current node α, how can we check the existence of a dominating node β? Fischetti and Toth answered this question by modeling the search of dominating nodes as a structured optimization problem, to be solved exactly or heuristically. For generic MIP models, this leads to the following *auxiliary problem*:

$$\begin{cases} XP^\alpha: & \min \sum_{j\in J^\alpha} c_j x_j \\ \text{s.t.} & \sum_{j\in J^\alpha} A_j x_j \le b^\alpha := \sum_{j\in J^\alpha} A_j x_j^\alpha \\ & x_j \text{ integer for all } j \in J^\alpha \end{cases} \tag{2.60}$$

If a solution x^β (say) of the auxiliary problem having a cost strictly smaller than $c(J^\alpha, x^\alpha)$ is found, then it defines a dominating node β and the current node α can be fathomed.

It is worth noting that the auxiliary problem is of the same nature as the original MIP problem, but with a smaller size and thus it is often easily solved (possibly in a heuristic way) by a general-purpose MIP solver, so we are indeed "MIPping the dominance test".

The Fischetti-Toth dominance procedure, called *Local Dominance* (LD) procedure in the sequel to stress its local nature, has several useful properties:

- there is no need to store any information about the set of previously generated nodes;
- there is no need to make any time-consuming comparison of the current node with other nodes;
- a node can be fathomed even if the corresponding dominating one has not been generated yet;
- the correctness of the enumerative algorithm does not depend on the branching rule; this is a valuable property since it imposes no constraints on the B&B parameters (though an unappropriate branching strategy could prevent several dominated nodes to be fathomed).

In addition, the LD test needs not be applied at every node. This is a crucial property from the practical point of view, as the dominance test introduces some overhead and it would make the algorithm uncompetitive if applied at every node. Note that skipping a LD test at a given node is not likely to induce a great pruning loss, since the following *inheritance* property holds (see [54] for the proof):

Proposition 1. *Let α and β be two nodes of the search tree and let β dominate α. Then for every α' successor of α there exists a node β' such that β' dominates α'.*

As a consequence, if β dominates α and α is not fathomed because the corresponding dominance test was skipped, we still have the possibility to prune some descendant nodes of α.

An important issue to be addressed when implementing the LD test is to avoid fathoming cycles arising when the auxiliary problem actually has a solution x^β different from x^α but of the same cost, in which case one is allowed to fathom node α only if a tie-break rule is used to guarantee that node β itself is not fathomed for the same reason. In order to prevent these "tautological" fathoming cycles the following criterion (among others) has been proposed in [30]: In case of cost equivalence, define as unfathomed the node β corresponding to the solution found by a *deterministic*[7] exact or heuristic algorithm used to solve the auxiliary problem. Unfortunately, this criterion can be misleading for two important reasons. First of all, it is not easy to define a "deterministic" algorithm for MIP. In fact, besides the possible effects of randomized steps, the output of the MIP solver typically depends, e.g., on the order in which the variables are listed on input, that can affect the choice of the branching variables as well as the internal heuristics.

In view of the considerations above, in our implementation we used a different tie-break rule, also described in [30], that consists in ranking cost-equivalent solutions in lexicographical order. To be more specific, in case of cost ties we fathom node α if and only if the partial solution x^β associated with the dominating node β is lexicographically smaller[8] than x^α. Using this tie-breaking rule, it is easy to prove [54] the correctness of the overall enumerative method.

2.4.1 Borrowing Nogoods from Constraint Programming

The computational overhead related to the LD test can be reduced considerably if we exploit the notion of nogoods taken from Constraint Programming. A *nogood* is a partial assignment of the problem variables such that every completion is either infeasible (for constraint satisfaction problems) or nonoptimal (for constraint optimization problems). The key observation here is that whenever we discover (through the solution of the auxiliary problem) that the current node α is dominated, we have indeed found a *nogood configuration* $[J^\alpha, x^\alpha]$ that we want to exclude from being re-analyzed at a later time.

There are two possible ways of exploiting nogoods in the context of MIP solvers:

[7] In our context, an algorithm is said to be deterministic if it always provides the same output solution for the same input set.

[8] We use the standard definition of lexicographic order on vectors of fixed size over a totally ordered set.

- Generate a constraint $\alpha^T x \leq \alpha_0$ cutting the nogood configuration off, so as to prevent it appears again in a later fractional solution. This is always possible (for both binary and general-integer linear problems) through a local branching constraint [25], and leads to the so-called *combinatorial Benders cuts* studied by Codato and Fischetti [14].
- Maintain explicitly a pool of previously found nogood configurations and solve the following problem (akin to separation) at each node α to be tested: Find, if any, a nogood configuration $[J', x']$ stored in the pool, such that $J' \subseteq J^\alpha$ and $x'_j = x^\alpha_j$ for all $j \in J'$. If the test is successful, we can of course fathom node α without the need of constructing and solving the auxiliary problem XP^α.

In our implementation we use the nogood-pool option, that according to our computational experience outperforms the cut options. It is worth noting that we are interested in minimal (with respect to set inclusion) nogoods, so as to improve both for efficiency and effectiveness of the method. Indeed, if node β dominates node α and $J' := \{j \in J^\alpha : x^\alpha_j \neq x^\beta_j\}$, then clearly the restriction of x^β onto J' dominates the restriction of x^α onto J'. If applied at every node, our procedure guarantees automatically the minimality of the nogood configurations found. If this is not the case, instead, minimality is no longer guaranteed, and is enforced by a simple post-processing step before storing any new nogood in the pool.

2.4.2 Improving the Auxiliary Problem

The effectiveness of the LD test presented in the previous section heavily depends on the auxiliary problem that is constructed at a given node α. In particular, it is crucial for its solution set to be as large as possible, so as to increase the chances of finding a dominating partial solution. Moreover, we aim at finding a partial solution different from (and hopefully lexicographically better than) the one associated with the current node; finding the same solution x^α is of no use within the LD context. For these reasons, several improvements of the original auxiliary-problem formulation have been proposed in [54], as outlined below.

The auxiliary problem XP^α constructed at node α is always feasible, as the partial assignment x^α corresponding to node α itself is always feasible. This is not a desired behavior, for two main reasons:

- Often x^α turns out to be the only feasible solution to XP^α—for our purposes, it would be better to consider it as infeasible, meaning that the node cannot be fathomed by our procedure.
- When solving the auxiliary problem, the solver often finds solution x^α (even if it is not provided explicitly on input for initializing the incum-

bent) and proves its optimality without looking for alternative (hopefully lexicographically better) optimal solutions.

Moreover, as the depth of the nodes in the B&B increases, the auxiliary problem grows in size and becomes heavier to solve. In addition, the resulting nogood (if any) may be of little applicability in the remaining part of the search because it may involve too many variables.

For these reasons one can heuristically limit the search space of the auxiliary problem to alternative assignments that are not too far from the current one, but different from it. This can be achieved again with two local branching [25] constraints, which however, in the most general case, could need the introduction of complicating auxiliary variables. According to our computational experience, a good compromise is to consider local branching constraints involving only the (binary or general integer) variables fixed to their lower or upper bound, namely:

$$\sum_{j \in U} (u_j - x_j) + \sum_{j \in L} (x_j - l_j) \leq k, \tag{2.61}$$

$$\sum_{j \in U} (u_j - x_j) + \sum_{j \in L} (x_j - l_j) \geq 1, \tag{2.62}$$

where

$$U = \{j \in J^\alpha \mid x_j^\alpha = u_j\} \quad \text{and} \quad L = \{j \in J^\alpha \mid x_j^\alpha = l_j\}.$$

It is worth noting that the above constraint may exclude some feasible solutions that differ from x^α with respect to variables fixed to values different from a lower or upper bound. In this case, our fathoming test can become less powerful, but the overall method remains correct. Finally, we found it useful to add the following *optimality* constraint

$$\sum_{j \in J^\alpha} c_j x_j \leq \sum_{j \in J^\alpha} c_j x_j^\alpha.$$

2.4.3 Computational Results

The enhanced dominance procedure presented in the previous section has been implemented in C++ within the `ILOG-Cplex` [41] framework on a Linux platform. Here are some implementation issues that deserve further description.

One of the main drawbacks of LD tests is that they postpone finding a better incumbent solution, thus increasing the number of nodes needed to solve the problem. This behavior is quite undesirable, especially in the first phase of the algorithm, when we have no incumbent and no nodes can be

fathomed through bounding criteria. A practical solution to this problem is to skip the dominance test until the first feasible solution is found.

The systematic application of the dominance test to every node of the search tree can become too heavy to be worthwhile in practice. A first consideration is that we should skip the dominance test on nodes near the top or the bottom of the search tree. Indeed, in the first case only a few variables have been fixed, hence there are little chances of finding a dominating partial assignment. In the latter case, instead, it is likely that the node would be pruned anyway by standard bounding tests; moreover, at the bottom of the tree the number of fixed variables is large and the auxiliary problem may be quite heavy to solve. In our implementation, we provide two thresholds on the tree depth of the nodes, namely $depth_{\min}$ and $depth_{\max}$, a node α being tested for dominance only if $depth_{\min} \leq \text{depth}(\alpha) \leq depth_{\max}$. Moreover, we decided to test for dominance a node only if its depth is a multiple of a given parameter, *depth_interval*. The three parameters above have been set as relative percentages on the number of variables. Finally, we set a limit on the computing time spent by the black-box MIP solver used for solving each auxiliary problem.

In our computational experiments we tested `ILOG-Cplex` 9.0 [41] commercial code with and without our LD test. All runs were performed on a AMD Athlon64 3500+ PC with 4GB of RAM, under Linux. The `ILOG-Cplex` code was run with its default options, and the overall time limit for processing each instance was set to 2,000 CPU seconds. As to LD tests, we used the following parameters:

- *depth_min* = 0.3 times the total number of variables;
- *depth_max* = 0.8 times the total number of variables;
- *depth_interval* = 0.1 times the total number of variables.

Moreover, in this implementation we did not use local branching constraint (2.61). The definition of the test-bed for testing the potentiality of our approach is a delicate issue. As a matter of fact, one cannot realistically expect any dominance relationship to be effective on all types of MIPs. Therefore, we looked for classes of problems whose structure can trigger the dominance relationship, and measured the speedup that can be achieved by using our specific LD procedure. In particular, we next give results on single and multiple knapsack problems [46]. We generated hard single knapsack instances according to the so-called *spanner instances* method in combination with the *almost strongly correlated* profit generation technique; see Pisinger [52] for details. Multiple knapsack instances were generated in a similar way, by choosing a same capacity for all the containers.

The results on hard single knapsack instances with 60 to 90 items are given in Table 2.7.

According to the table, LD tests are very effective on this class of problems, yielding consistently a large speedup and solving to optimality three instances where the standard code reached the time limit. It is worth noting that little

Table 2.7 Computational results for hard single knapsack instances.

	Standard Cplex			Dominance Code			Ratio	
Problem	Nodes	Time (s)	Gap	Nodes	Time (s)	Gap	Nodes	Time
kp60_1	286,056	9.60	0	725	0.04	0	394.56	240.00
kp60_2	27,108,819	2,050.82	0.016	773,890	2,067.44	0.028	35.03	1.00
kp60_3	718,887	24.89	0	1330	0.34	0	540.52	73.21
kp60_4	804,304	26.32	0	28,947	12.17	0	27.79	2.16
kp60_5	688,122	24.36	0	48,895	4.87	0	14.07	5.00
kp70_1	23,671,129	2,050.52	0.406	1,638,641	2,047.53	0.406	14.45	1.00
kp70_2	1,060,259	35.43	0	153,552	61.54	0	6.90	0.58
kp70_3	665,668	23.12	0	147,899	28.00	0	4.50	0.83
kp70_4	23,037,172	2,048.61	0.399	935,986	2,065.34	0.216	24.16	1.00
kp70_5	424,815	15.17	0	19,685	0.89	0	21.58	17.04
kp80_1	413,489	13.98	0	249,582	10.39	0	1.66	1.35
kp80_2	587,456	22.54	0	140,191	7.25	0	4.19	3.11
kp80_3	673,318	22.61	0	26,803	2.13	0	25.12	10.62
kp80_4	529,026	17.56	0	5,274	0.24	0	100.31	73.17
kp80_5	32,604,432	2,050.79	0.328	460,908	109.26	0	70.74	18.77
kp90_1	25,409,911	2,047.65	0.065	928,586	2,034.52	0.065	27.36	1.00
kp90_2	37,650,100	2,041.93	0.137	3,957,332	167,81	0	9.51	12.17
kp90_3	3,024,346	126.59	0	266,137	16.82	0	11.36	7.53
kp90_4	1,926,498	81.39	0	134,385	10.25	0	14.34	7.94
kp90_5	26,510,264	2,052.64	0.263	1,047,483	551.93	0	25.31	3.72
Total	207,794,071	14,787.34	-	10,966,231	9,198.76	-	18.95	1.61

Table 2.8 Parameter tuning on specific hard single knapsack instances.

Problem	Nodes	Time (s)	Gap	Depth Min	Depth Max
kp60_2	1,653,691	68.24	0	0.3	0.6
kp70_2	7,763	1.02	0	0.3	0.8
kp80_2	11,171	0.57	0	0.3	0.8
kp80_3	2,955	0.30	0	0.3	0.6

parameter tuning would have produced better results in terms of elapsed time and/or final gap for four instances, as in Table 2.8.

As to hard multiple knapsack problems, we have generated instances with a number of items ranging from 20 to 40 and a number of knapsacks ranging form 3 to 5. The LD parameters were set to:

- $depth\_min = 0.3$ times the total number of variables;
- $depth\_max = 0.5$ times the total number of variables;
- $depth\_interval = 0.1$ times the total number of variables.

For these problems, the time limit was increased to one hour.

The results on multiple knapsack problems are available in Table 2.9.

In the multiple knapsack case, LD was not as effective as in the single case, yielding some improvements only on the smallest instances. It is how-

Table 2.9 Computational results for hard multiple knapsack problems.

Problem	Standard Cplex			Dominance Code			Ratio	
	Nodes	Time (s)	Gap	Nodes	Time (s)	Gap	Nodes	Time
mkp20_3.lp	187,713	16.13	0	129,516	12.18	0	1.45	1.32
mkp20_4.lp	80,845	11.20	0	79,650	10.20	0	1.02	1.10
mkp20_5.lp	1,639,251	171.07	0	1,563,925	181.96	0	1.05	0.94
mkp30_3.lp	33,205,268	3,652.63	0.627	31,141,581	3,654.65	0.632	1.07	1.00
mkp30_4.lp	5,414,529	650.24	0	26,707,752	3,649.55	0.439	0.20	0.18
mkp30_5.lp	28,159,141	3,644.38	0.515	25,653,218	3,649.69	0.515	1.10	1.00
mkp40_3.lp	40,576,080	3,654.31	0.515	22,963,265	3,699.83	0.515	1.77	0.99
mkp40_4.lp	25,354,639	3,645.51	0.437	21,250,331	3,652.06	0.437	1.19	1.00
mkp40_5.lp	777,810	160.81	0	389,427	128.98	0	2.00	1.25
Total	135,395,276	15,606.28	-	129,878,665	18,639.10	-	1.04	0.84

ever worth mentioning that the ratio *dominated_nodes*/*dominance_tests* was quite good also for these problems (though lower than in the single knapsack case) and that the effectiveness of LD could have been hidden by the limited computational time given to the solvers.

Acknowledgements This work was supported by the Future and Emerging Technologies unit of the EC (IST priority), under contract no. FP6-021235-2 (project "ARRIVAL") and by MiUR, Italy.

References

1. T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4:77–86, 2007.
2. T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34:361–372, 2006. Problems available at *http://miplib.zib.de*.
3. E. Amaldi, M.E. Pfetsch, and L.E. Trotter Jr. On the maximum feasible subsystem problem, IISs and IIS-hypergraphs. *Mathematical Programming*, 95:533–554, 2003.
4. E. Balas, S. Ceria, M. Dawande, F. Margot, and G. Pataki. OCTANE: A new heuristic for pure 0–1 programs. *Operations Research*, 49:207–225, 2001.
5. E. Balas and C.H. Martin. Pivot-and-complement: A heuristic for 0-1 programming. *Management Science*, 26:86–96, 1980.
6. E. Balas and M. Perregaard. Lift-and-project for mixed 0-1 programming: Recent progress. *Discrete Applied Mathematics*, 123:129–154, 2002.
7. E. Balas and A. Saxena. Optimizing over the split closure. *Mathematical Programming*, 113:219–240, 2008.
8. E. Balas, S. Schmieta, and C. Wallace. Pivot and shift — a mixed integer programming heuristic. *Discrete Optimization*, 1:3–12, 2004.
9. R.E. Bixby, S. Ceria, C.M. McZeal, and M.W.P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.
10. P. Bonami, G. Cornuéjols, S. Dash, M. Fischetti, and A. Lodi. Projected Chvátal–Gomory cuts for mixed integer linear programs. *Mathematical Programming*, 113:241–257, 2008.

11. A. Caprara and A.N. Letchford. On the separation of split cuts and related inequalities. *Mathematical Programming*, 94:279–294, 2002.
12. J.W. Chinneck. Fast heuristics for the maximum feasible subsystem problem. *INFORMS Journal on Computing*, 13:210–223, 2001.
13. V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 5:305–337, 1973.
14. G. Codato and M. Fischetti. Combinatorial Benders cuts. In D. Bienstock and G. Nemhauser, editors, *Integer Programming and Combinatorial Optimization, IPCO X*, volume 3064 of *Lecture Notes in Computer Science*, pages 178–195. Springer, 2004.
15. W. Cook, R. Kannan, and A. Schrijver. Chvatal closures for mixed integer programming problems. *Mathematical Programming*, 47:155–174, 1990.
16. G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112:3–44, 2008.
17. G. Cornuéjols and Y. Li. On the rank of mixed 0,1 polyhedra. *Mathematical Programming*, 91:391–397, 2002.
18. E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102:71–90, 2005.
19. S. Dash, O. Günlük, and A. Lodi. On the MIR closure of polyhedra. In M. Fischetti and D.P. Williamson, editors, *Integer Programming and Combinatorial Optimization, IPCO XII*, volume 4513 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2007.
20. S. Dash, O. Günlük, and A. Lodi. MIR closures of polyhedral sets. *Mathematical Programming*, DOI 10.1007/s10107-008-0225-x, 2008.
21. Double-Click sas. personal communication, 2001.
22. J. Eckstein and M. Nediak. Pivot, cut, and dive: a heuristic for 0-1 mixed integer programming. *Journal of Heuristics*, 13:471–503, 2007.
23. F. Eisenbrand. On the membership problem for the elementary closure of a polyhedron. *Combinatorica*, 19:297–300, 1999.
24. M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104:91–104, 2005.
25. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.
26. M. Fischetti and A. Lodi. MIPping Closures: An instant survey. *Graphs and Combinatorics*, 23:233–243, 2007.
27. M. Fischetti and A. Lodi. Optimizing over the first Chvátal closure. *Mathematical Programming*, 110:3–20, 2007.
28. M. Fischetti and A. Lodi. Repairing MIP infeasibility through local branching. *Computers & Operations Research*, 35:1436–1445, 2008.
29. M. Fischetti, C. Polo, and M. Scantamburlo. A local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. *Networks*, 44:61–72, 2004.
30. M. Fischetti and P. Toth. A New Dominance Procedure for Combinatorial Optimization Problems. *Operations Research Letters*, 7:181–187, 1988.
31. J. Gleeson and J. Ryan. Identifying minimally infeasible subsystems of inequalities. *ORSA Journal on Computing*, 2:61–63, 1990.
32. F. Glover and M. Laguna. General purpose heuristics for integer programming – part I. *Journal of Heuristics*, 2:343–358, 1997.
33. F. Glover and M. Laguna. General purpose heuristics for integer programming – part II. *Journal of Heuristics*, 3:161–179, 1997.
34. F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
35. R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.
36. R.E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The Rand Corporation, 1960.

37. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.
38. P. Hansen, N. Mladenović, and D. Urosevic. Variable neighborhood search and local branching. *Computers & Operations Research*, 33:3034–3045, 2006.
39. F.S. Hillier. Efficient heuristic procedures for integer linear programming with an interior. *Operations Research*, 17:600–637, 1969.
40. T. Ibaraki, T. Ohashi, and F. Mine. A heuristic algorithm for mixed-integer programming problems. *Mathematical Programming Study*, 2:115–136, 1974.
41. ILOG S.A. *CPLEX: ILOG CPLEX 11.0 User's Manual and Reference Manual*, 2007. `http://www.ilog.com`.
42. G.W. Klau. personal communication, 2002.
43. A. Løkketangen. Heuristics for 0-1 mixed-integer programming. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*, pages 474–477. Oxford University Press, 2002.
44. A. Løkketangen and F. Glover. Solving zero/one mixed integer programming problems using tabu search. *European Journal of Operational Research*, 106:624–658, 1998.
45. F. Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94:71–90, 2002.
46. S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, New York, 1990.
47. A.J. Miller. personal communication, 2003.
48. N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.
49. J.L. Nazareth. The homotopy principle and algorithms for linear programming. *SIAM Journal on Optimization*, 1:316–332, 1991.
50. G. Nemhauser and L. Wolsey. A recursive procedure to generate all cuts for 0-1 mixed integer programs. *Mathematical Programming*, 46:379–390, 1990.
51. C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.
52. D. Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32:2271–2284, 2005.
53. E. Rothberg. personal communication, 2002.
54. D. Salvagnin. A dominance procedure for integer programming. Master's thesis, University of Padua, October 2005.

Chapter 3
MetaBoosting: Enhancing Integer Programming Techniques by Metaheuristics

Jakob Puchinger, Günther R. Raidl, and Sandro Pirkwieser

Abstract This chapter reviews approaches where metaheuristics are used to boost the performance of exact integer linear programming (IP) techniques. Most exact optimization methods for solving hard combinatorial problems rely at some point on tree search. Applying more effective metaheuristics for obtaining better heuristic solutions and thus tighter bounds in order to prune the search tree in stronger ways is the most obvious possibility. Besides this, we consider several approaches where metaheuristics are integrated more tightly with IP techniques. Among them are collaborative approaches where various information is exchanged for providing mutual guidance, metaheuristics for cutting plane separation, and metaheuristics for column generation. Two case studies are finally considered in more detail: (i) a Lagrangian decomposition approach that is combined with an evolutionary algorithm for obtaining (almost always) proven optimal solutions to the knapsack constrained maximum spanning tree problem and (ii) a column generation approach for the periodic vehicle routing problem with time windows in which the pricing problem is solved by local search based metaheuristics.

3.1 Introduction

When considering optimization approaches that combine aspects from metaheuristics with mathematical programming techniques, the resulting hybrid

Jakob Puchinger
arsenal research, Vienna, Austria
e-mail: jakob.puchinger@arsenal.ac.at

Günther R. Raidl · Sandro Pirkwieser
Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria
e-mail: {raidl,pirkwieser}@ads.tuwien.ac.at

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_3,

system may either be of *exact* or *heuristic* nature. Exact approaches are guaranteed to yield proven optimal solutions when they are given enough computation time. In contrast, heuristics only aim at finding reasonably good approximate solutions usually in a more restricted time; performance guarantees are typically not provided. Most of the existing hybrid approaches are of heuristic nature, and mathematical programming techniques are used to boost the performance of a metaheuristic. Exploiting solutions to exactly solvable relaxations of the original problem, or searching large neighborhoods by means of mathematical programming techniques are examples for such approaches; see also Chapter 4. On the other hand, there are also several highly successful ways to exploit metaheuristic strategies for enhancing the performance of mathematical programming techniques, and often these methods retain their exactness. We refer to such improvement techniques as *MetaBoosting* and study them in detail in the present chapter.

Most exact approaches for solving hard *combinatorial optimization problems* (COPs) are based on a *tree search*, where the search space is recursively partitioned in a divide-and-conquer manner into mutually disjoint subspaces by fixing certain variables or imposing additional constraints. In a naive enumeration tree each subspace is further divided as long as it contains more than one feasible solution. Obviously, the size of such a naive search tree increases rapidly with the problem size, and naive enumeration is therefore inefficient. The key to successfully approach larger problem instances is to have some mechanism for substantially pruning the search tree. This is usually done by identifying subspaces that need not to be further pursued, as they cannot contain a feasible solution that is better than a solution already found before. The scalability of a tree search thus depends essentially on the efficiency of this pruning mechanism.

In *branch-and-bound* (B&B), upper and lower bounds are determined for the objective values of solutions, and subspaces for which the lower bounds exceed the upper bounds are discarded. Considering a minimization problem, any feasible solution provides a (global) upper bound. Thus, any (meta-) heuristic that is able to determine good heuristic solutions in reasonable time may be an essential help in B&B for pruning the search tree, even when the heuristic itself does not provide any performance guarantee.

Applying an effective metaheuristic to obtain better upper bounds for B&B is the most obvious way how one can boost the performance of an exact optimization technique by means of a metaheuristic. When considering established *integer (linear) programming* techniques including cutting plane methods, column generation, and diverse variants of relaxation based approaches in more detail, we can observe several further possibilities for exploiting the strengths of metaheuristics.

The next section will introduce our basic notations and briefly review important IP techniques. In Sections 3.3 to 3.5 we describe various successful MetaBoosting strategies. Two exemplary case studies are presented together with some practical results in more detail in Sections 3.6 and 3.7. First, we

consider a Lagrangian decomposition/evolutionary algorithm hybrid for the knapsack constrained maximum spanning tree problem, and second, a column generation approach that uses metaheuristics for solving the pricing problem is discussed for the periodic vehicle routing problem with time windows. Conclusions are drawn in Section 3.8.

3.2 Integer Programming Techniques

This section introduces some basic notations and gives a short introduction into prominent IP techniques. For an in-depth coverage of the subject we refer to the books on linear optimization by Bertsimas and Tsitsiklis [6] and on combinatorial and integer optimization by Nemhauser and Wolsey [37] and Wolsey [53].

We consider IP problems of the form

$$z_{\text{IP}} = \min\{cx \mid Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}, \tag{3.1}$$

where x is an n-dimensional integer variable vector in column form and $c \in \mathbb{Q}^n$ an n-dimensional row vector. Their dot-product cx is the *objective function* that should be minimized. Matrix $A \in \mathbb{Q}^{m \times n}$ and the m-dimensional column vector $b \in \mathbb{Q}^m$ together define m inequality constraints. A *mixed integer program* (MIP) would involve a combination of integer and real-valued variables.

Maximization problems can be transformed into minimization problems by simply changing the sign of c. Less-than constraints are similarly brought into greater-than-or-equal form by changing the sign of the corresponding coefficients, and equalities can be translated to pairs of inequalities. Thus, we can handle all kinds of linear constraints by appropriate transformations. Without loss of generality, we may therefore restrict our following considerations to minimization problems of this standard form.

3.2.1 Relaxations and Duality

One of the most important concepts in integer programming are *relaxations*, where some or all constraints of a problem are loosened or omitted. Relaxations are mostly used to obtain related, simpler problems that can be solved efficiently yielding bounds and approximate (not necessarily feasible) solutions for the original problem. Embedded within a B&B framework, these techniques may lead to effective exact solution techniques.

The *linear programming* (LP) relaxation of the IP (3.1) is obtained by relaxing the integrality constraints, yielding

$$z_{\text{LP}} = \min\{cx \mid Ax \geq b, x \geq 0, x \in \mathbb{R}^n\}. \tag{3.2}$$

Large instances of such LPs can be efficiently solved using simplex-based or interior-point algorithms. The solution to the LP relaxation provides a lower bound for the original minimization problem, i.e. $z_{\text{IP}} \geq z_{\text{LP}}$, since the search space of the IP is contained within the one of the LP and the objective function remains the same.

We can further associate a *dual problem* to an LP (3.2), which is defined by

$$w_{\text{LP}} = \max\{ub \mid uA \leq c, u \geq 0, u \in \mathbb{R}^m\} \tag{3.3}$$

with u being the m-dimensional dual variable row vector. The dual of the dual LP is the original (*primal*) LP again. Important relations between the primal problem and its dual are known as weak and strong duality theorems, respectively:

- **Weak duality theorem:** The value of every finite feasible solution to the dual problem is a lower bound for the primal problem, and each value of a finite feasible solution to the primal problem is an upper bound for the dual problem. As a consequence, if the dual is unbounded, the primal is infeasible and vice versa.
- **Strong duality theorem:** If the primal has a finite optimal solution with value z^*_{LP}, than its dual has the same optimal solution value $w^*_{\text{LP}} = z^*_{\text{LP}}$ and vice versa.

In case of an IP we have to distinguish between weak and strong duals: A *weak dual* of an IP (3.1) is any maximization problem $w = \max\{w(u) \mid u \in S_{\text{D}}\}$ such that $w(u) \leq cx$ for all $x \in \{Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}$. An obvious weak dual of (3.1) is the dual (3.3) of its LP relaxation (3.2). A *strong dual* is a weak dual that further has an optimal solution u^* such that $w(u^*) = cx^*$ for an optimal solution x^* of (3.1). For solving IPs, weak duals which are iteratively strengthened during the course of the optimization process are often utilized.

Another commonly used relaxation of IPs, which often yields significantly tighter bounds than the LP relaxation, is *Lagrangian relaxation* [20, 21]. Consider the IP

$$z_{\text{IP}} = \min\{cx \mid Ax \geq b, Dx \geq d, x \geq 0, x \in \mathbb{Z}^n\}, \tag{3.4}$$

where constraints $Ax \geq b$ are "easy" in the sense that the problem can be efficiently solved when the m' "complicating" constraints $Dx \geq b$ are dropped. Simply removing these constraints yields a relaxation, but the resulting bound will usually be weak because of this complete ignorance. In Lagrangian relaxation, constraints $Dx \geq d$ are replaced by corresponding penalty terms in the objective function:

$$z_{\text{LR}}(\lambda) = \min\{cx + \lambda(d - Dx) \mid Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}. \tag{3.5}$$

Vector $\lambda \in \mathbb{R}^{m'}$ is the vector of Lagrangian multipliers, and for any $\lambda \geq 0$, $z_{\mathrm{LR}}(\lambda) \leq z_{\mathrm{IP}}$, i.e., we have a valid relaxation of the IP. We are now interested in finding a specific vector λ yielding the best, i.e. largest, possible lower bound, which leads to the *Lagrangian dual problem*

$$z^*_{\mathrm{LR}} = \max_{\lambda \geq 0} \{z_{\mathrm{LR}}(\lambda)\}. \tag{3.6}$$

This Lagrangian dual is a piecewise linear, convex function which can usually be well solved by iterative procedures like a subgradient method. A more elaborate algorithm that has been reported to converge faster on several problems is the volume algorithm [4], whose name is inspired by the fact that primal solutions are also considered, whose values come from approximating the volumes below active faces of the dual problem.

Given a solution λ to the Lagrangian dual problem (3.6) and a corresponding optimal solution x^* to the Lagrangian relaxation (3.5) that is also feasible to the original problem (3.4), i.e. $Dx^* \geq d$, the following complementary slackness condition holds: x^* is an optimal solution to the original problem (3.4) if and only if

$$\lambda(d - Dx^*) = 0. \tag{3.7}$$

Provided that the Lagrangian dual problem is solved to optimality, it can be shown that the Lagrangian relaxation always yields a bound that is at least as good as the one of the corresponding linear relaxation.

A third general-purpose relaxation technique for IPs is *surrogate relaxation* [26]. Here, some or all constraints are scaled by surrogate multipliers and cumulated into a single inequality by adding the coefficients. Similarly as in Lagrangian relaxation, the ultimate goal is to find surrogate multipliers yielding the overall best bound. Unfortunately, this surrogate dual problem usually has not such nice properties as the Lagrangian dual problem and solving it is often difficult. However, if one is able to determine optimal surrogate multipliers, the bound obtained for the IP is always at least as good as (and often better than) those obtained from the corresponding linear and Lagrangian relaxations.

3.2.2 LP-Based Branch-and-Bound

By solving the LP relaxation of an IP we obtain a lower bound on the optimal IP solution value and the solution in general will contain fractional variable values. (If all variable values would be integer, we already would have solved the IP.) The standard way to continue towards an optimal integer solution is the already mentioned B&B. Branching usually takes place over some variable x_i with a fractional LP-value x^*_i, defining as first subproblem the IP with

the additional inequality $x_i \leq \lfloor x_i^* \rfloor$ and as second subproblem the IP with inequality $x_i \geq \lceil x_i^* \rceil$. For these subproblems with the additional branching constraints, the LP relaxations are resolved leading to increased lower bounds and eventually solutions where all integer variables have integral values. As mentioned in the introduction, primal heuristics are usually also applied to each subproblem in order to find improved feasible solutions and corresponding global upper bounds, enabling a stronger pruning of the search tree.

3.2.3 Cutting Plane Algorithm and Branch-and-Cut

When modeling COPs as IPs an important goal is to find a *strong* formulation, for which the solution value of the LP relaxation in general provides a *tight* bound. For many COPs it is possible to strengthen an existing IP formulation significantly by including further inequalities, which would actually be redundant w.r.t. the integer optimum. In general it is even possible to strengthen a model such that the LP relaxation already yields an integer optimum. However, the number of required constraints often grows exponentially with the problem size. Naively solving such an LP by standard techniques might quickly become too costly in practice.

Dantzig et al. [10] proposed the *cutting plane algorithm* for this purpose, which usually only considers a fraction of all constraints explicitly but is nevertheless able to determine an optimal solution to the whole LP.

The cutting plane approach starts by solving a reduced LP consisting only of a small subset of initial inequalities. It then tries to find inequalities that are violated by the obtained solution but are valid for the original problem (i.e. contained in the full LP). These valid inequalities are called *cuts* or *cutting planes*, and they are added to the current reduced LP, which is then resolved. The whole process is iterated until no further cutting planes can be determined. If the algorithm computing the cuts provides a proof that no further violated inequality exists, the final solution is optimal for the original full LP. The subproblem of identifying cuts is called *separation problem*. In practice it is crucial to have an efficient method for separating cuts as usually a significant number of valid inequalities must be derived until the cutting plane algorithm terminates.

From a theoretical point of view it is possible to solve any IP using a pure cutting plane approach with appropriate classes of cuts. There exist generic types of cuts, such as the Chvatal-Gomory cuts [53], which guarantee such a result. In practice, however, it may take a too long time for such a cutting plane approach to converge to the optimum, partly because it is often a hard subproblem to separate effective cuts and partly because of the large number of needed cuts.

The combination of B&B with cutting plane methods yields the highly effective class of *branch-and-cut* algorithms which are widely used. Specialized

branch-and-cut approaches have been described for many applications and are known for their effectiveness. Cut separation is usually applied at each node of the B&B tree to tighten the bounds of the LP relaxation and to exclude infeasible solutions as far as possible.

For cutting plane separation effective heuristic methods come into play once again: For strengthening the LP relaxations it is often sufficient to generate cuts heuristically since the correctness of the final solution does not depend on the generated cuts as long as they are valid. Almost all modern MIP solvers include sophisticated generic cut separation heuristics, and they play a major role in the success of these solvers.

3.2.4 Column Generation and Branch-and-Price

Often it is possible to model COPs via strong formulations involving a huge number of variables. Dantzig-Wolfe decomposition [11] is a technique for obtaining such models from compact formulations in a systematic way. It replaces the original problem variables by linear combinations of the extreme points and extreme rays of the original search space, yielding a potentially exponential number of new variables. The obtained models can result in much stronger relaxations than their compact counterparts.

Despite the many variables, the LP relaxations of such formulations can often be efficiently calculated. The *column generation* approach starts with only a small subset of all variables (corresponding to columns in the matrix notation of the IP) and solves the corresponding restricted LP relaxation. It is then tried to identify one or more so far ignored variables whose inclusion may lead to an improved solution. This subproblem is called *pricing problem*. For a minimization problem a variable can eventually improve the current LP solution if it has negative reduced costs. After adding such a new variable to the restricted LP, it is resolved and the process iterated until no further variables with negative reduced costs exist. The final solution is an optimal solution for the complete LP.

Column generation can be seen as dual to the cutting plane approach, since inequalities correspond to variables in the dual LP. For a recent review on column generation see [35]. The cutting stock problem is an early example for the successful application of column generation based methods [24]. Every possible cutting pattern is represented by a variable and the pricing problem corresponds to the classical 0–1 knapsack problem, which can be solved efficiently in pseudo-polynomial time.

As the column generation algorithm only solves the LP relaxation, it must in general also be combined with B&B in order to obtain optimal integer solutions. When column generation is performed for each node of the B&B tree, the approach is called *branch-and-price*. One of the main difficulties in the implementation of such methods lies in the development of appropriate

branching rules. Furthermore, the individual LPs may sometimes be degenerated, or newly added columns may only improve the solutions marginally leading to many iterations until convergence. In the latter cases, stabilization techniques as discussed in [13] often improve the situation.

Similarly as cutting plane separation may be performed by effective heuristics, one can also heuristically solve the pricing problem in column generation. Care must be taken that in the final iteration it is necessary to prove that no further columns with negative reduced costs exist so that the obtained solution value is guaranteed to be a lower bound for the original IP.

Finally, it occasionally makes sense to combine a cutting plane approach with column generation and embed both in B&B. Such methods, called *branch-and-cut-and-price*, are sometimes extremely successful but are typically also rather complex and highly specialized.

3.3 Metaheuristics for Finding Primal Bounds

Branch-and-bound based approaches rely on tight primal bounds that are most commonly obtained from feasible solutions. Obviously, heuristics and metaheuristics can be applied to the original problem before starting the B&B process, providing initial solutions. The search space of the exact method is immediately reduced, usually improving overall computation times. Such an approach has the practical advantage of also providing feasible solutions at an early stage of the optimization process.

Furthermore (meta-)heuristics can be repeatedly applied throughout the whole tree search, providing possibly improved solutions. Again, this can speed up the overall optimization essentially by further pruning the search tree. Even the optimal solution might be discovered by one of those heuristics. On the other hand, when heuristics are applied too often and have rather long run-times, they might slow down the overall process. Thus, an appropriate balance is required.

3.3.1 Initial Solutions

Generic MIP based heuristics for computing initial solutions are widely used. They range from early heuristics such as described in [2, 30] over pivot and complement [3] to the recent feasibility pump [17, 5], which is also discussed in Chapter 2 of this book. The major commercial generic MIP solvers such as CPLEX[1] or XPRESS MP[2] have very strong heuristics for finding initial

[1] http://www.ilog.com

[2] http://www.dashoptimization.com

feasible solutions, often outperforming simple problem-specific heuristics in terms of solution quality and speed. Unfortunately, not much is publicly known about these heuristics.

An interesting approach specifically tailored to the *multidimensional knapsack problem* (MKP) involving metaheuristics is presented in Vimont et al. [52]. The MKP can be defined by the following IP:

$$\text{(MKP)} \qquad \text{maximize} \quad z = \sum_{j=1}^{n} p_j x_j \tag{3.8}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \quad i = 1, \ldots, m, \tag{3.9}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{3.10}$$

A set of n items with profits $p_j > 0$ and m resources with capacities $c_i > 0$ are given. Each item j consumes an amount $w_{ij} \geq 0$ from each resource i. Variables x_j indicate which items are selected. The objective is to choose a subset of items with maximum total profit that does not violate any of the capacity constraints (3.9).

An exact algorithm based on implicit enumeration and reduced cost propagation is applied. The enumeration algorithm tries to first handle the unpromising parts of the search space, with the goal of reducing it substantially. After computing an initial solution yielding a lower bound, the problem is first partitioned by fixing the number of selected items to certain values [50]. Each of the resulting subproblems is then explored by B&B with a special branching strategy based on the solution to the LP relaxation and reduced costs at each search tree node.

The search space is further reduced by fixing some variables using a propagation mechanism. It is based on the reduced cost constraint originally described in [38]. After solving the LP relaxation yielding a solution ($\overline{x}$), the following reduced cost inequality can be devised:

$$\sum_{j:\overline{x}_j=0} |\overline{c}_j| x_j + \sum_{j:\overline{x}_j=1} |\overline{c}_j|(1 - x_j) \leq \text{UB} - \text{LB}, \tag{3.11}$$

where $\overline{c}$ is the reduced cost vector corresponding to $\overline{x}$ and LB is a primal lower bound, typically the objective value of a feasible solution.

This approach relies heavily on tight primal bounds, since constraint (3.11) becomes tighter with increasing values of LB. These bounds come from a sophisticated tabu search based hybrid algorithm described in [50]. The search space is partitioned via additional constraints fixing the total number of items to be packed. Lower and upper bounds for the number of items are calculated by solving modified LP relaxations of the original MKP. For each remaining partition of the search space, tabu search is independently applied, starting with a solution derived from the LP relaxation of the partial problem. The

whole tabu search approach has further been improved in [51] by additional variable fixing.

This example demonstrates that a combination of highly developed specialized methods for computing bounds with the aid of a metaheuristic, generating dependent cuts, and guiding the search is sometimes able to achieve exceedingly good results.

3.3.2 B&B Acting as Local Search Based Metaheuristic

Fischetti and Lodi proposed *local branching* as an extension for generic branch-and-cut based MIP solvers with the aim of producing good heuristic solutions early during the exact tree search [19]. Local branching introduces the spirit of classical k-opt local search in B&B by modifying the branching rule and the strategy for choosing the next tree node to process. Let us consider MIPs with 0–1 variables; let $x = (x_1, \ldots, x_n)$ be the variable vector and $\mathcal{B} \subseteq \{1, \ldots, n\}$ be the index set of the 0–1 variables. A k-opt neighborhood around a given incumbent solution $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_n)$ can be defined by the *local branching constraint*

$$\Delta(x, \overline{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k, \tag{3.12}$$

where $\overline{S}$ corresponds to the index set of the 0–1 variables that are set to one in the incumbent solution, i.e., $\overline{S} = \{j \in \mathcal{B} \mid \overline{x}_j = 1\}$. $\Delta(x, \overline{x})$ resembles the classical Hamming distance between x and $\overline{x}$ for integer values.

Starting from an initial solution, the search space is partitioned into the k-opt neighborhood of this incumbent and the remaining part of the search space by applying the local branching constraint and its inverse $\Delta(x, \overline{x}) \geq k + 1$, respectively. The MIP solver is then forced to find the best solution in the k-opt neighborhood first. If an improved solution x' has been found, a new subproblem $\Delta(x, \overline{x}')$ corresponding to the search of the k-opt neighborhood of this new incumbent is split off the remaining search space and solved in the same way; otherwise a larger k may be tried. The process is repeated until no further improvement can be achieved. Finally, the remaining problem corresponding to all yet unconsidered parts of the search space is processed in a standard way.

This basic mechanism is extended by introducing time limits, automatically modifying the neighborhood size k, and adding diversification strategies to improve performance. An extension of the branching constraint for general integer variables is also described. Results on various MIP benchmark instances using CPLEX as MIP solver indicate the advantages of the approach in terms of an earlier identification of high quality solutions.

Hansen et al. [27] suggest a variant of local branching which follows more closely the classical variable neighborhood search metaheuristic for choosing the next k-opt neighborhood to process. Improved results are reported. Fischetti et al. [18] describe another variant of the original local branching where they consider problems in which the set of variables naturally partitions into two levels and fixing the first-level variables to some values yields substantially easier subproblems.

Danna et al. [9] suggest a different approach called *relaxation induced neighborhood search* (RINS) for exploring the neighborhoods of incumbent solutions more intensively. The central idea is to occasionally devise a sub-MIP at a node of the B&B tree that corresponds to a special neighborhood of an incumbent solution: Variables having the same values in the incumbent and in the current solution of the LP relaxation are fixed, and an objective cutoff is set based on the objective value of the incumbent. A sub-MIP is solved on the remaining variables with a given time limit. If a better solution can be found it is passed to the global MIP-search, which is resumed after the sub-MIP's termination. In the authors' experiments, CPLEX is used as MIP solver, and RINS is compared to standard CPLEX, local branching, combinations of RINS and local branching, and guided dives. Results indicate that RINS often performs best. CPLEX includes RINS as a standard strategy for quickly obtaining good heuristic solutions since version 10. Local branching constraints are said to be often less effective as they are dense inequalities involving all integer variables. In particular, adding the inverse local branching constraints of already searched k-opt neighborhoods to the remaining problem is found to be disadvantageous as the reduced node processing throughput caused by the series of these dense constraints outweighs the benefit of avoiding redundant exploration of parts of the search space.

Recently Ghosh [23] proposed a *distance induced neighborhood search* (DINS). It is conjectured that better MIP solutions are more likely to be close to the solution of the LP relaxation than farther away. Hence, an appropriate distance metric is utilized. DINS combines soft fixing of variables as in local branching as well as hard fixing of variables as in RINS, plus an additional rebounding procedure, which adapts the lower and upper bounds of selected variables. Experimental results indicate that DINS outperforms both local branching and RINS; DINS is also integrated now in CPLEX.

3.3.3 Solution Merging

In *solution merging* new, possibly better solutions are created from attributes appearing in two or more promising heuristic solutions. Such an approach is based on the assumption that high quality solutions often share many attributes.

Recombination, the primary variation operator in genetic algorithms, can be seen as a classical solution merging approach. Usually, two parent solutions are selected and an offspring is derived by simple random inheritance of parental attributes. Classical recombination operations do not try to optimize this offspring, which therefore often is worse than its parents. However, these operations are computationally cheap and can be repeated many times in order to achieve improvements.

Alternatively, one can put more effort into the derivation of such offspring. A sometimes effective technique is *path relinking* [25], which traces a path in the search space from one parent to a second by repeatedly exchanging a single attribute only (or more generally by performing a series of moves in a simple neighborhood structure). An overall best solution found on this path is finally taken as offspring.

This idea can further be extended by considering not just solutions on a single path between two parents, but the whole subspace of solutions induced by the joined attributes appearing in a set of two or more input solutions. An *optimal merging* operation returns a best solution from this subspace, i.e., it identifies a best possible combination of the parents' attributes. Depending on the underlying problem, identifying such an optimal offspring is often a hard optimization problem on its own, but due to the usually quite limited number of different attributes appearing in the parents, it can often be solved in reasonable time in practice.

For mixed integer programming, Rothberg [47] suggests a tight integration of an evolutionary algorithm (EA) including optimal merging in a branch-and-cut based MIP solver. In regular intervals the EA algorithm is applied as B&B tree node heuristic. The population of the EA consists of the best non-identical solutions found so far, which have either been discovered by the MIP tree search or by previous iterations of the EA.

Mutation selects one parent, fixes a randomly chosen subset of variables, and calls the MIP solver for determining optimal values for the remaining problem. Since the number of variables to be fixed is a critical parameter, an adaptive scheme is applied to control it. In contrast to classical EAs, mutation is performed before recombination on a fixed number of randomly chosen solutions, since at the beginning of the optimization only one or very few solutions will be in the population.

Recombination is performed by first fixing all variables that have the same values in two selected parental solutions and applying the MIP solver to this reduced subproblem. The exploration of this subproblem is eventually truncated when a given node-limit is exceeded. New high quality solutions discovered during this search are added to the population. This recombination is further generalized to more than two parents by fixing variable values that are identical in all of them.

The applied selection strategy simply chooses the first parent from the population at random, and the second is then chosen randomly amongst the solutions with a better objective value than the first one. This guarantees a

certain bias towards better solutions. For mutation the same mechanism is used, but only the second solution is used.

Experimental results indicate that this hybrid often is able to find significantly better solutions than other heuristic methods for several very difficult MIPs. The method is integrated in the commercial MIP solver CPLEX since version 10.

3.3.4 Metaheuristics and Lagrangian Relaxation

As mentioned in Section 3.2.1, Lagrangian relaxations may sometimes yield substantially tighter lower bounds than simpler LP relaxations. Furthermore, heuristic solutions and, thus, upper bounds are often either automatically obtained as intermediate by-products from the subgradient procedure or by applying typically rather simple Lagrangian heuristics such as rounding or repairing procedures. When embedded in a B&B framework, such Lagrangian relaxation based methods are frequently turned into highly successful exact optimization approaches.

To further improve performance by obtaining better upper bounds, more sophisticated metaheuristics may be applied in combination with Lagrangian relaxation. For example, a well-working hybrid of a Lagrangian relaxation approach and variable neighborhood descent has recently been described for a real-world fiber optic network design problem in Leitner and Raidl [33].

An interesting additional aspect of such combinations is that also the metaheuristic may benefit by exploiting diverse intermediate results from the subgradient search. A successful example for this is the hybrid Lagrangian GA for the prize collecting Steiner tree problem proposed by Haouari and Siala [28]. They apply a Lagrangian relaxation on a minimum spanning tree formulation of the prize collecting Steiner tree problem and use the volume algorithm for solving the Lagrangian dual. After termination, the GA is started on a reduced problem, consisting only of the edges appearing in all the intermediate trees derived by the volume algorithm. Furthermore, some of the GA's initial solutions are derived from the volume algorithm's intermediate reduced edge costs by applying a greedy Lagrangian heuristic. Last but not least, the GA uses a modified objective function: Instead of the original costs, the reduced costs that are finally obtained by the volume algorithm are used; in this way, the metaheuristic search is guided into regions of the search space deemed promising by the Lagrangian relaxation.

The authors of the present chapter describe a similar approach for the knapsack constrained maximum spanning tree problem in [40]. Section 3.6 summarizes this work as an exemplary case study.

3.4 Collaborative Hybrids

In collaborative combinations of different types of optimization techniques, the algorithms exchange information but are not part of each other; i.e., there is no clear master containing the other method(s) as subprocedures [42]. The individual algorithms may be executed sequentially, intertwined, or in a parallel way and exchange information for guidance. In principle, any metaheuristic that provides incumbent solutions to a B&B-based approach might already be considered to fall into this class of approaches. The above mentioned hybrid Lagrangian relaxation approach from Haouari and Siala can, e.g., also be regarded a sequential collaborative combination, where the Lagrangian relaxation provides guidance for the GA.

Intertwined and parallel combinations allow for *mutual* guidance, i.e., all participating methods may exploit information from each other. Talukdar et al. [49] describe a very general agent-based model for such systems, called *asynchronous teams* (A-Teams). This problem solving architecture consists of a collection of agents and memories connected in a strongly cyclic directed way, and each optimization agent works on the target problem, a relaxation, or a subclass of the original problem. Denzinger and Offerman [12] describe a similar framework called TECHS (TEams for Cooperative Heterogeneous Search). It consists of teams of one or more agents using the same search paradigm. Communication between the agents is controlled by so-called send- and receive-referees.

A specific example for a successful intertwined collaboration of an EA and the branch-and-cut based MIP solver XPRESS MP is the hybrid algorithm from French et al. [22] for solving general IPs. It starts with a branch-and-cut phase, in which information from the B&B tree nodes is collected in order to derive candidate solutions that are added to the originally randomly initialized EA-population. When a certain criterion is satisfied, the EA takes over for some time using the augmented initial population. After termination of the EA, its best solutions are passed back and grafted onto the B&B tree. Full control is given back to branch-and-cut after the newly added nodes had been examined to a certain degree. Reported results on instances of the maximum satisfiability problem show that this hybrid yields better solutions than XPRESS MP or the EA alone.

Another cooperative approach involving a memetic algorithm and branch-and-cut has been described by Puchinger et al. [44] for the MKP. Both methods are performed in parallel and exchange information in a bidirectional asynchronous way. In addition to promising primal solutions, the memetic algorithm also receives dual variable values of certain LP relaxations and uses them for improving its repair and local improvement functions by updating the items' pseudo-utility ratios. Results that are often better than those from [50] and partly competitive to those from [51] have been obtained.

3.5 Metaheuristics for Cut and Column Generation

As already pointed out in Section 3.2, in cut and column generation based IP methods the dynamic separation of cutting planes and the pricing of columns can be done by means of (meta-)heuristics in order to speed up the optimization process. Such approaches are reviewed in more detail in the following two sections.

3.5.1 Cut Separation

In branch-and-cut algorithms inequalities that are satisfied by feasible integer solutions but are violated by the current solution to the LP relaxation have to be derived quickly. Of course, the cuts one wants to find should be *strong* in the sense that they cut away "large" portions of the search space, leading to a significant increase of the LP solution value and thus to relatively few iterations until convergence of the cutting plane algorithm. As many classes of strong cuts are difficult to separate, heuristic separation procedures are commonly applied. More sophisticated metaheuristics, however, have so far only rarely been used for this purpose. A reason might be the usually large number of cuts that must be generated, and hence the strong requirements w.r.t. speed. Nevertheless, there exist some examples of successful metaheuristic cut separation approaches.

Augerat et al. [1] consider a capacitated vehicle routing problem and describe a branch-and-cut algorithm in which a sequence of methods consisting of a simple construction heuristic, a randomized greedy method, and a tabu search is used for separating capacity constraints. The approach starts with the fastest simple heuristic and switches to the next, more complex strategy as long as no valid cutting plane could be found.

Another example is the branch-and-cut algorithm by Gruber and Raidl for the bounded diameter minimum spanning tree problem described in detail in Chapter 8 of this book. The diameter bound is ensured via an exponentially large number of so-called jump inequalities. Again, a sequence of methods is used for their separation, starting from a greedy construction technique over a local search procedure to a tabu search algorithm. On several benchmark instances, this algorithm outperforms other state-of-the-art IP approaches for this problem, and some larger instances than before could be solved to proven optimality.

Rei et al. [46] describe the acceleration of Benders decomposition by local branching. The basic principle of Benders decomposition is to project a MIP into the space of complicating integer variables only; continuous variables and the constraints involving them are replaced by corresponding constraints on the integer variables. These constraints, however, are not directly available but need to be dynamically created. According to the classical method, an

optimal solution to the relaxed master problem (including only the already separated cuts) is needed and an LP involving this solution must be solved in order to separate a single new cut. Rei et al. improved this method by introducing phases of local branching on the original problem in order to obtain multiple feasible heuristic solutions. These solutions provide improved upper bounds and further allow to derive multiple additional cuts before the relaxed master problem needs to be resolved.

3.5.2 Column Generation

In column generation based algorithms the pricing problem often is difficult by itself, and applying fast (meta-)heuristics can be a meaningful option. It can be beneficial for the overall performance if most of the columns are heuristically derived.

Filho and Lorena [16] apply a heuristic column generation approach to graph coloring. A GA is used to generate initial columns and to solve the pricing problem at every iteration. Column generation is performed as long as the GA finds columns with negative reduced costs. The master problem is solved using CPLEX.

Puchinger and Raidl [41, 43] describe an exact branch-and-price algorithm for the three-stage two-dimensional bin packing problem. Rectangular items have to be orthogonally packed into the least number of larger rectangles of fixed size, and only non-overlapping three-stage guillotine packing patterns are allowed. The pricing problem occurring in this application is a three-stage two-dimensional knapsack packing problem. Fast column generation is performed by applying a sequence of four methods: (i) a greedy heuristic, (ii) an evolutionary algorithm, (iii) solving a restricted, simpler IP-model of the pricing problem using CPLEX within a certain time-limit, and finally (iv) solving a complete IP-model by CPLEX. The algorithms coming later in this sequence are only executed if the previous ones did not find columns with negative reduced costs. The greedy heuristic is based on the classical finite first fit heuristic but is adapted to consider additional constraints introduced by the branching decisions during the search process of the branch-and-price algorithm. The EA uses a direct set-based representation for solutions making it possible to ignore the order of the items to be packed and therefore avoiding redundancies introduced by many symmetries. Specific recombination and mutation operators were developed for this problem. The presented computational experiments show that each pricing algorithm contributes essentially to the whole column generation process. Applied to large problem instances with limited run-time, better solutions are often obtained by the sequential pricing compared to using just one strategy. It is conjectured that also in other applications such combinations of multiple (meta-)heuristic and exact pricing algorithms may be beneficial.

3.6 Case Study: A Lagrangian Decomposition/EA Hybrid

This first case study demonstrates a combination of a Lagrangian decomposition approach with an EA for the knapsack constrained maximum spanning tree problem. The EA exploits information of the Lagrangian decomposition and improves previously obtained primal solutions. Proven optimal solutions are obtained in most cases, especially also on large problem instances. More details on this work can be found in [40].

3.6.1 The Knapsack Constrained Maximum Spanning Tree Problem

The *knapsack constrained maximum spanning tree* (KCMST) problem arises in practical situations where the aim is to design a most profitable communication network under a strict limit on total costs, e.g. for cable laying or similar resource constraints. The problem is also referred to as budget or side constrained minimum spanning tree problem and is NP-hard [54].

It is defined on an undirected connected graph $G = (V, E)$ with node set V and edge set $E \subseteq V \times V$ representing all possible connections. Each edge $e \in E$ has associated a weight $w_e \in \mathbb{Z}_+$ (corresponding to costs) and a profit $p_e \in \mathbb{Z}_+$. In addition, a weight limit (capacity) $c > 0$ is specified. We seek a spanning tree $G_T = (V, T)$, $T \subseteq E$, on G that maximizes the total profit $\sum_{e \in T} p_e$ and where weight $\sum_{e \in T} w_e$ does not exceed c. By introducing binary variables x_e, $\forall e \in E$, indicating which edges are part of the solution, i.e. $x_e = 1 \leftrightarrow e \in T$ and $x_e = 0$ otherwise, the problem can be stated as:

$$\text{(KCMST)} \quad \max \quad p(x) = \sum_{e \in E} p_e x_e \tag{3.13}$$

$$\text{s.t.} \quad x \text{ represents a spanning tree on } G, \tag{3.14}$$

$$\sum_{e \in E} w_e x_e \leq c, \tag{3.15}$$

$$x_e \in \{0, 1\}, \qquad \forall e \in E. \tag{3.16}$$

Obviously, the problem represents a combination of the classical minimum spanning tree problem (with changed sign in the objective function) and the 0–1 knapsack problem due to constraint (3.15). Yamada et al. [54] proposed a straight-forward Lagrangian relaxation where the knapsack constraint is relaxed and primal solutions are improved by local search. We enhance this approach in the following.

3.6.2 Lagrangian Decomposition of the KCMST Problem

The aforementioned natural combination lends itself to obtain tighter upper bounds via *Lagrangian decomposition* (LD), which is a special variant of Lagrangian relaxation that can be meaningful when there is evidence of two or possibly more intertwined subproblems, and each of them can be efficiently solved on its own by specialized algorithms.

For this purpose, we duplicate variables x_e, $\forall e \in E$, by introducing new, corresponding variables y_e and including linking constraints, leading to the following reformulation:

$$\max \quad p(x) = \sum_{e \in E} p_e x_e \tag{3.17}$$

$$\text{s.t.} \quad x \text{ represents a spanning tree on } G, \tag{3.18}$$

$$\sum_{e \in E} w_e y_e \leq c, \tag{3.19}$$

$$x_e = y_e, \qquad \forall e \in E, \tag{3.20}$$

$$x_e, y_e \in \{0,1\}, \qquad \forall e \in E. \tag{3.21}$$

Now we relax the linking constraints (3.20) in a Lagrangian fashion using Lagrangian multipliers $\lambda_e \in \mathbb{R}$, $\forall e \in E$, hence obtaining the Lagrangian decomposition of the original problem, denoted by KCMST-LD(λ):

$$\max \quad p(x) = \sum_{e \in E} p_e x_e - \sum_{e \in E} \lambda_e (x_e - y_e) \tag{3.22}$$

$$\text{s.t.} \quad x \text{ represents a spanning tree on } G, \tag{3.23}$$

$$\sum_{e \in E} w_e y_e \leq c, \tag{3.24}$$

$$x_e, y_e \in \{0,1\}, \qquad \forall e \in E. \tag{3.25}$$

Stating KCMST-LD(λ) in a more compact way and emphasizing the now independent subproblems yields

$$\text{(MST)} \quad \max \ \{(p-\lambda)^T x \mid x \hat{=} \text{ a spanning tree on } G,\ x \in \{0,1\}^E\} \ + \tag{3.26}$$

$$\text{(KP)} \quad \max \ \{\lambda^T y \mid w^T y \leq c,\ y \in \{0,1\}^E\}. \tag{3.27}$$

For a given λ, the maximum spanning tree (MST) subproblem (3.26) can be efficiently solved by standard algorithms. The 0–1 knapsack subproblem (3.27) is known to be weakly NP-hard and we apply the COMBO dynamic programming algorithm [36] for efficiently solving it.

To obtain the tightest (smallest) upper bound, we have to solve the Lagrangian dual problem:

$$\min_{\lambda \in \mathbb{R}^E} v(\text{KCMST-LD}(\lambda)), \tag{3.28}$$

where $v(\text{KCMST-LD}(\lambda))$ denotes the optimal solution value to KCMST-LD(λ). This is achieved by applying the volume algorithm [4].

3.6.3 Lagrangian Heuristic

We employ several methods to also derive heuristic solutions and corresponding lower bounds. An obvious Lagrangian heuristic is the following: Whenever the spanning tree created in an iteration of the volume algorithm satisfies the capacity limit, we already have a feasible KCMST. In order to further improve such solutions we consecutively apply a local search based on an edge exchange neighborhood. Thereby we select an edge (u, v) not present in the solution and identify the least profitable edge—choosing an edge with highest weight in case of ties—of the path that connects nodes u and v in the current tree and that may be replaced by (u, v) without violating the capacity constraint. We then exchange these two edges in case the profit increases or it stays the same but the overall weight decreases. The edge to be included, (u, v), is either chosen (i) at random from $E \setminus T$, or (ii) at the beginning of the local search, all edges are sorted according to decreasing $p'_e = p_e - \lambda_e$ (the reduced profits used to solve the MST subproblem) and in every iteration of the local search the next less profitable edge not active in the current solution is chosen. The latter selection scheme results in a greedy search where every edge is considered at most once. Since Lagrangian multipliers are supposed to be of better quality in later phases of the optimization process, local search is only applied when the ratio of the incumbent lower and upper bounds is larger than a certain threshold τ. Local search stops after 100 consecutive non-improving iterations have been performed.

3.6.4 Evolutionary Algorithm for the KCMST

The EA for heuristically solving the KCMST is based on a direct edge-set representation as described in [45]. This encoding and its corresponding variation operators are known to provide strong locality and heritability, and all operations can efficiently be performed in time that depends (almost) only linearly on the number of nodes.

The general framework is steady-state, i.e., in each iteration one feasible offspring solution is created by means of recombination, mutation, and eventually local improvement, and it replaces the worst solution in the population. Duplicates are not allowed in the population; they are always immediately discarded. The EA's operators work as follows.

Initialization: A diversified initial population is obtained via a random spanning tree construction based on Kruskal's algorithm with a bias towards selecting edges with high profits. In case a generated solution is infeasible with respect to the knapsack constraint, it is stochastically repaired by iteratively selecting a not yet included edge at random, adding it to the tree, and removing an edge with highest weight from the induced cycle.

Recombination: An offspring is derived from two selected parental solutions in such a way that it always exclusively consists of inherited edges: In a first step all edges contained in both parents are immediately adopted. The remaining ones are merged into a single candidate list. From this list, we iteratively select edges by binary tournaments with replacement favoring high-profit edges again. Selected edges are included in the solution if they do not introduce a cycle; otherwise, they are discarded. The process is repeated until a complete spanning tree is obtained. If it exceeds the capacity limit, the solution is repaired in the same way as during initialization, but only considering parental edges for inclusion.

Mutation: Mutation is performed by inserting a new randomly selected edge and removing another edge from the introduced cycle. The choice of the edge to be included is again biased towards high-profit edges by utilizing a normally-distributed rank-based selection, see [45]. The edge to be removed from the induced cycle is chosen at random among those edges whose removal retains a feasible solution.

Local Search: With a certain probability, a newly derived candidate solution is further improved by the previously described local search procedure.

3.6.5 LD/EA Hybrid

For the LD/EA hybrid we apply similar ideas as described in [28] for the prize collecting Steiner tree problem, where the EA is used successfully for finding better final solutions after performing LD. Here, the EA is adapted to exploit a variety of (intermediate) results from LD. Of course, the EA is only applied if the best feasible solution obtained by LD does not correspond to the determined upper bound; otherwise a proven optimal solution is already found. These steps are performed after LD has terminated and before the EA is executed:

1. For the selection of edges during initialization, recombination, and mutation, original edge profits p_e are replaced by reduced profits $p'_e = p_e - \lambda_e$. In this way, Lagrangian dual variables are exploited, and the heuristic search emphasizes the inclusion of edges that turned out to be beneficial in LD.
2. The edge set to be considered by the EA is reduced from E to a subset E' containing only those edges that appeared in any of the feasible solutions encountered by LD. For this purpose, LD is extended to mark those edges.

3. The best feasible solution obtained by LD is directly included in the EA's initial population.
4. Finally, the upper bound obtained by LD is exploited by the EA as an additional stopping criterion: When a solution with a corresponding total profit is found, it is optimal, and the EA terminates.

3.6.6 Experimental Results

The ability of the LD to yield extremely tight upper bounds that are significantly better than those resulting from the simple Lagrangian relaxation [54] is documented in [40]. Here we concentrate on the ability of the involved heuristics for improving the primal solutions. Therefore, we show and compare results for the pure Lagrangian decomposition (LD), LD with local search (LD+LS), and the LD/EA hybrid (LD+LS+EA). Due to the absence of publicly available test instances we generated maximal planar graphs ($\mathrm{P}_{|V|,\gamma}$), and random ($\mathrm{R}_{|V|,|E|,\gamma,\delta}$) as well as complete graphs ($\mathrm{K}_{|V|,\gamma,\delta}$) as detailed in [29]. The instances differ in

1. size: number of nodes $|V|$ and edges $|E|$,
2. profit/weight correlation γ: being uncorrelated, weakly or strongly correlated for maximal planar graphs and of type outliers, weakly or strongly correlated for the other graph types,
3. and capacity limit δ: low, medium, or high limit.

A detailed treatment of these instances is given in [40]. For the optional local search, greedy edge selection is used for random and complete graphs with an application threshold set to $\tau = 0.99$ and random edge selection with $\tau = 0.995$ for the maximal planar graphs. The EA operates with a population size of 100 individuals, binary tournament selection is used. Local search is applied with a probability of 20% on each new candidate solution. The maximum number of EA iterations is 10000 for maximal planar graphs and 30000 for random and complete graphs. The edge set reduction was applied only in case of maximal planar graphs, as it turned out to be sometimes too restricting in the other cases.

All experiments were performed on a 2.2 GHz AMD Athlon 64 PC with 2 GB RAM. The results are given in Table 3.1; ten runs per instance were performed for the stochastic algorithms. We state the CPU-time in seconds $t[s]$, the number of iterations *iter*, the average lower bounds (LB), i.e., the objective values of the best feasible solutions. Upper bounds (UB) are expressed in terms of the relative gap to these lower bounds: $gap = (UB - LB)/LB$; corresponding standard deviations are listed in columns σ_{gap}. Columns *%-Opt* show the percentages of instances for which the gaps are zero and, thus, optimality has been achieved. For LD+LS+EA, the table additionally lists the average numbers of EA iterations $iter_{\mathrm{EA}}$, the relative amounts of edges

discarded after performing LD $red = (|E| - |E'|)/|E| \cdot 100\%$, stating (*red*) in case no reduction was applied, and the percentages of optimal solutions $\%\text{-}Opt_{\text{EA}}$, among $\%\text{-}Opt$, found by the EA.

As can be seen, the solutions obtained by LD are already quite good and gaps are small in general. Applying the local search (LD+LS) always improves the average lower bound and in some cases helps to find more proven optimal solutions, which in turn reduces the number of iterations of the volume algorithm. The hybrid approach (LD+LS+EA) further boosts the average solution quality in almost all cases and substantially increases the number of solutions for which optimality could be proven; the increase in running time one has to pay is mostly only moderate. Of course, in order to solve the very few remaining instances to proven optimality as well, one could embed LD+LS+EA within a B&B.

3.7 Case Study: Metaheuristic Column Generation

In this section we discuss as a second case study a successful application of metaheuristics for solving the pricing subproblem within a column generation approach. The presented results are part of a currently ongoing project of the authors.

3.7.1 The Periodic Vehicle Routing Problem with Time Windows

Periodic vehicle routing problems (PVRPs) are generalized variants of the classical vehicle routing problem (VRP) where customersmust be served several times within a given planning period. They occur in real-world applications as in courier services, grocery distribution or waste collection. The PVRP considered here is the *Periodic Vehicle Routing Problem with Time Windows* (PVRPTW). It is defined on a complete directed graph $G = (V, A)$, where $V = \{v_0, v_1, \dots v_n\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in V,\, i \neq j\}$ is the arc set. The planning horizon shall be t days, also referred to as $T = \{1, \dots, t\}$. Vertex v_0 represents the depot with time window $[e_0, l_0]$ at which we have a fleet of m homogeneous vehicles with capacity Q and maximal daily working time D. Each vertex $i \in V_C$, with $V_C = V \setminus \{v_0\}$, corresponds to a customer and has an associated demand $q_i \geq 0$, a service duration $d_i \geq 0$, a time window $[e_i, l_i]$, a service frequency f_i and a set C_i of allowable combinations of visit days. For each arc $(v_i, v_j) \in A$ there are given travel times (costs) $c_{ij} \geq 0$. The aim is (i) to select a single visit combination per customer and (ii) to find at most m vehicle routes on each of the t days on G, such that

Table 3.1 Results of Lagrangian decomposition and hybrid algorithms on maximal planar, random, and complete graphs.

Instance	LD						LD+LS						LD+LS+EA							
	$t[s]$	$iter$	LB	gap $[\cdot 10^{-5}]$	σ_{gap} $[\cdot 10^{-5}]$	$\%$-Opt	$t[s]$	$iter$	LB	gap $[\cdot 10^{-5}]$	σ_{gap} $[\cdot 10^{-5}]$	$\%$-Opt	$t[s]$	red	$iter_{\text{EA}}$	LB	gap $[\cdot 10^{-5}]$	σ_{gap} $[\cdot 10^{-5}]$	$\%$-Opt	$\%$-Opt_{EA}
$P_{2000,u}$	1.48	791	147799.50	0.0683	0.2049	90	2.28	782	147799.55	0.0342	0.1489	95	2.90	41.21	150	147799.60	0	0	100	5
$P_{2000,w}$	1.52	853	85570.50	0.3519	0.7513	80	2.38	844	85570.63	0.1994	0.5261	86	4.26	42.61	457	85570.78	0.0235	0.1643	98	12
$P_{2000,s}$	2.12	1030	82521.70	1.9389	2.3118	40	2.66	868	82523.30	0	0	100	2.66	21.99	0	82523.30	0	0	100	0
$P_{4000,u}$	3.35	859	294872.00	0.0340	0.1019	90	5.59	841	294872.03	0.0238	0.0866	93	8.64	40.17	316	294872.10	0	0	100	7
$P_{4000,w}$	4.19	1053	170956.70	0.8195	0.9155	40	6.15	978	170957.79	0.1813	0.306	72	14.66	43.82	842	170958.06	0.0234	0.1147	96	24
$P_{4000,s}$	4.71	1066	165049.80	1.0300	0.8590	30	5.99	915	165051.44	0.0364	0.1439	94	9.95	19.92	410	165051.48	0.0121	0.0848	98	4
$P_{6000,u}$	5.66	912	441977.80	0.0680	0.1038	70	9.33	886	441977.96	0.0317	0.0786	86	15.41	40.25	339	441978.10	0	0	100	14
$P_{6000,w}$	6.55	1022	256317.40	0.3904	0.4621	50	9.25	964	256318.09	0.1210	0.2452	76	24.47	45.14	909	256318.36	0.0156	0.0764	96	20
$P_{6000,s}$	8.14	1157	247587.90	1.7368	1.3032	20	10.44	996	247592.04	0.0646	0.1481	84	33.73	19.94	1401	247592.09	0.0444	0.1264	89	5
$P_{8000,u}$	8.32	960	589446.50	0.1017	0.1357	60	13.81	918	589446.89	0.0356	0.077	81	28.44	39.98	595	589447.09	0.0017	0.0168	99	18
$P_{8000,w}$	9.78	1107	341902.50	0.5555	0.5139	30	14.18	1037	341903.85	0.1609	0.2124	58	48.40	44.82	1384	341904.37	0.0088	0.0499	97	39
$P_{8000,s}$	10.88	1125	330117.10	1.5147	1.3065	20	14.20	990	330121.86	0.0727	0.1294	76	57.00	17.99	1727	330121.96	0.0424	0.1051	86	10
$R_{300,11213,o,l}$	9.53	1737	542839.40	1.7477	1.8326	10	11.72	1737	542840.60	1.5271	1.5937	10	29.99	(92.93)	27000	542843.63	0.9706	0.6928	10	0
$R_{300,11213,o,m}$	7.10	1536	580716.50	0.2583	0.2464	30	8.89	1506	580716.60	0.2411	0.2576	40	21.43	(91.63)	18000	580716.64	0.2342	0.2477	40	0
$R_{300,11213,o,h}$	3.57	1260	591409.00	0.1690	0.2507	50	5.11	1259	591409.30	0.1183	0.1320	50	13.73	(91.02)	12285	591409.54	0.0778	0.1132	64	14
$R_{300,11213,s_2,l}$	24.58	1563	77466.60	8.5209	5.6046	20	24.45	1409	77473.00	0.2581	0.5161	80	24.69	(80.64)	336	77473.20	0	0	100	20
$R_{300,11213,s_2,m}$	15.37	1351	155244.80	5.4064	5.1165	0	14.77	1051	155253.20	0	0	100	14.73	(81.54)	0	155253.20	0	0	100	0
$R_{300,11213,s_2,h}$	16.52	1332	232877.70	6.5305	5.2668	10	16.74	1238	232892.50	0.1718	0.2847	70	18.34	(85.28)	2222	232892.89	0.0043	0.0428	99	29
$R_{300,22425,o,l}$	26.39	3324	568771.90	6.8383	6.1475	10	32.10	3324	568788.80	3.8714	4.3327	10	52.08	(95.24)	26700	568796.00	2.6042	3.3654	11	1
$R_{300,22425,o,m}$	14.70	1943	588410.30	0.2210	0.2020	30	18.83	1943	588410.50	0.1870	0.1605	30	33.05	(95.46)	18078	588410.80	0.1360	0.1272	40	10
$R_{300,22425,o,h}$	7.28	1358	594373.50	0.0168	0.0505	90	10.10	1358	594373.50	0.0168	0.0505	90	12.40	(94.54)	3000	594373.50	0.0168	0.0505	90	0
$R_{300,22425,s_2,l}$	44.08	2059	77445.70	12.2628	9.0170	0	42.58	1793	77455.20	0	0	100	42.58	(86.26)	0	77455.20	0	0	100	0
$R_{300,22425,s_2,m}$	29.69	1687	154940.30	7.8185	8.9007	10	28.81	1392	154952.40	0	0	100	28.81	(93.71)	0	154952.40	0	0	100	0
$R_{300,22425,s_2,h}$	34.63	1964	232424.80	16.2741	12.5659	10	36.55	1885	232461.90	0.3013	0.3874	50	44.59	(89.39)	10682	232462.37	0.0990	0.1811	77	27
$K_{300,o,l}$	247.29	19163	582646.00	4.0334	7.1749	10	316.33	19163	582660.30	1.5789	1.4435	10	333.98	(97.50)	27000	582663.46	1.0366	0.8511	10	0
$K_{300,o,m}$	40.44	2909	592797.70	0.1856	0.1401	30	45.96	2864	592797.90	0.1518	0.1401	40	55.19	(97.70)	10212	592798.50	0.0506	0.0773	70	30
$K_{300,o,h}$	30.13	2373	596076.40	0.0503	0.1074	80	35.49	2371	596076.50	0.0336	0.0671	80	36.13	(96.94)	1239	596076.70	0	0	100	20
$K_{300,s_2,l}$	63.20	2495	77225.70	28.6269	20.8442	0	60.80	2195	77247.80	0	0	100	60.80	(93.07)	0	77247.80	0	0	100	0
$K_{300,s_2,m}$	62.25	2704	154445.00	12.4958	8.3394	0	59.11	2404	154464.30	0	0	100	59.11	(94.48)	0	154464.30	0	0	100	0
$K_{300,s_2,h}$	76.60	3396	231665.00	15.9285	18.7408	10	78.10	3142	231701.90	0	0	100	78.10	(92.77)	0	231701.90	0	0	100	0

1. each route starts and ends at the depot,
2. each customer i belongs to f_i routes over the planning horizon,
3. the total demand of the route for each vehicle does not exceed the capacity limit Q, and its duration does not exceed the maximal working time D,
4. the service at each customer i begins in the interval $[e_i, l_i]$ and every vehicle leaves the depot and returns to it in the interval $[e_0, l_0]$, and
5. the total travel costs of all vehicles are minimized.

We further assume so-called hard time windows, i.e., arriving before e_i at customer i incurs a waiting time at no additional costs, whereas arriving later than l_i is not allowed. The PVRPTW has been first mentioned in Cordeau et al. [8], where a tabu search metaheuristic is described for it.

3.7.2 Set Covering Formulation for the PVRPTW

Among the most successful solution approaches for VRPs in general are algorithms based on column generation. Therefore, we focus on an IP formulation suitable for such an approach and formulate the integer master problem (IMP) for the PVRPTW as a set covering model:

$$\min \sum_{\tau \in T} \sum_{\omega \in \Omega} \gamma_\omega \, v_{\omega\tau} \tag{3.29}$$

$$\text{s.t.} \quad \sum_{r \in C_i} y_{ir} \geq 1, \qquad \forall i \in V_C, \tag{3.30}$$

$$\sum_{\omega \in \Omega} v_{\omega\tau} \leq m, \qquad \forall \tau \in T, \tag{3.31}$$

$$\sum_{\omega \in \Omega} \alpha_{i\omega} \, v_{\omega\tau} - \sum_{r \in C_i} \beta_{ir\tau} \, y_{ir} \geq 0, \qquad \forall i \in V_C, \forall \tau \in T, \tag{3.32}$$

$$y_{ir} \in \{0,1\}, \qquad \forall i \in V_C, \forall r \in C_i, \tag{3.33}$$

$$v_{\omega\tau} \in \{0,1\}, \qquad \forall \omega \in \Omega, \forall \tau \in T. \tag{3.34}$$

The set of all feasible individual routes is denoted by Ω, and with each route $\omega \in \Omega$ we have associated costs γ_ω and variables $v_{\omega\tau}$, $\forall \tau \in T$, representing the number of times route ω is selected on day τ. For each customer $i \in V_C$, variable y_{ir} indicates whether or not visit combination $r \in C_i$ is chosen. The objective is to minimize the total costs of all routes (3.29). Covering constraints (3.30) guarantee that at least one visit day combination is selected per customer, fleet constraints (3.31) restrict the number of daily routes to not exceed the number of available vehicles m, and visit constraints (3.32) link the routes and the visit combinations, whereas $\alpha_{i\omega}$ and $\beta_{ir\tau}$ are binary constants indicating if route ω visits customer i and if day τ belongs to visit combination $r \in C_i$ of customer i, respectively.

3.7.3 Column Generation for Solving the LP Relaxation

Here, our aim is to derive a lower bound for the IMP by exactly solving its LP relaxation. An extension of the approach towards an exact branch-and-price algorithm is part of our ongoing work. Conditions (3.33) and (3.34) are replaced by $y_{ir} \geq 0$ and $v_{\omega\tau} \geq 0$, yielding the (linear) master problem (MP). Due to the large number of variables (columns) corresponding to routes, this LP cannot be solved directly. Instead, we restrict ourselves to a small number of initial columns $\Omega' \subset \Omega$, yielding the corresponding restricted master problem (RMP). Additional columns (routes) that are able to improve the current LP solution are generated by iteratively solving the pricing subproblem, which resembles in our case a *shortest path problem with resource constraints* (SPPRC) [31] and is NP-hard. Regarding the quality of the theoretically obtainable lower bound it is beneficial to restrict the search to elementary paths, hence only considering the *elementary SPPRC* (ESPPRC). The following ESPPRC pricing subproblem holds for each day $\tau \in T$ and is solved on an auxiliary graph $G' = (V', A')$, with $V' = V \cup \{v_{n+1}\}$ and $A' = \{(v_0, i), (i, v_{n+1}) : i \in V_C\} \cup \{(i, j) : i, j \in V_C,\, i \neq j\}$, where v_{n+1} is a copy of the (starting) depot v_0 and acts as target node:

$$\min \sum_{i \in V'} \sum_{j \in V'} \hat{c}_{ij\tau}\, x_{ij} \tag{3.35}$$

$$\text{s.t.} \quad \sum_{j \in V_C} x_{0j} = 1 \tag{3.36}$$

$$\sum_{i \in V'} x_{ik} - \sum_{j \in V'} x_{kj} = 0 \qquad \forall k \in V_C \tag{3.37}$$

$$\sum_{i \in V_C} x_{i,n+1} = 1 \tag{3.38}$$

$$\sum_{i \in V_C} \sum_{j \in V'} q_i\, x_{ij} \leq Q \tag{3.39}$$

$$a_{n+1} - w_0 \leq D \tag{3.40}$$

$$a_i + w_i + d_i + c_{ij} - M_{ij}(1 - x_{ij}) \leq a_j \qquad \forall (i, j) \in A' \tag{3.41}$$

$$e_i \leq (a_i + w_i) \leq l_i \qquad \forall i \in V' \tag{3.42}$$

$$w_i \geq 0 \qquad \forall i \in V' \tag{3.43}$$

$$a_i \geq 0 \qquad \forall i \in V' \setminus \{v_0\} \tag{3.44}$$

$$a_0 = 0 \tag{3.45}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A' \tag{3.46}$$

Variables x_{ij}, $\forall (i,j) \in A'$, denote which arcs from A' are used, and $\hat{c}_{ij\tau}$ are the reduced costs of using arc (i,j) on day τ:

$$\hat{c}_{ij\tau} = \begin{cases} c_{ij} - \rho_\tau & \text{if } i = v_0,\ j \in V_C, \\ c_{ij} - \pi_{i\tau} & \text{if } i \in V_C,\ j \in V'. \end{cases} \tag{3.47}$$

with ρ_τ and $\pi_{i\tau}$ being the dual variable values of constraints (3.31) and (3.32), respectively. Equalities (3.36) to (3.38) are flow conservation constraints, and inequalities (3.39) and (3.40) guarantee to not exceed the capacity and duration limits, respectively. Finally, (3.41) and (3.42) are time constraints, with variable a_i denoting the arrival time at customer i and w_i the waiting time occurring after a_i.

3.7.4 Exact and Metaheuristic Pricing Procedures

We apply an exact algorithm as well as metaheuristics for solving the ESPPRC subproblem. The former is realized by a dynamic programming approach based on [7, 14]. We use a label correcting algorithm and expand the partial paths from the depot v_0 to the target node v_{n+1}, thereby retaining only non-dominated labels taking into account the cumulated costs, load, duration, and overall waiting time, as well as the arrival time and the set of unreachable nodes. To minimize route duration we adhere to the concept of *forward time slack* [48] and maximize the waiting time w_0 at the depot without introducing a time window violation. This is also considered when extending labels and checking the dominance relation. The algorithm can also be stopped after a certain number of negative cost paths have been found, i.e., applying a "forced early stop", c.f. [32].

The first metaheuristic is an instance of iterated local search (ILS) [34]. It starts with the "empty" path (v_0, v_{n+1}) with zero costs and applies in each iteration a perturbation and a subsequent local improvement phase. Both phases make use of the following neighborhood structures: inserting, deleting, moving, replacing, and exchanging individual customers. The local search selects them in a random fashion and always accepts the first improving change. Perturbation applies ten random neighborhood moves in a sequence.

Our second, alternative metaheuristic approach can be regarded a greedy randomized adaptive search procedure (GRASP) [15]: In each iteration we start with the "empty" path (v_0, v_{n+1}) with zero costs and successively try to add arcs having negative costs, always selecting one at random in case there are more available; afterwards we also apply the perturbation and local search phase as described for the ILS algorithm.

Whenever an iteration of the metaheuristics results in a negative cost path it is stored and returned at the end of the procedure. Once one or more negative cost routes have been determined for one of the daily subproblems,

corresponding variables are priced in for all days and the RMP is resolved. In the following iteration we start the column generation with the same daily subproblem before considering the others. The whole process is continued until a full iteration over all days yields no new negative cost routes.

3.7.5 Experimental Results

Benchmark instances were taken from [8]. They are divided in types 'a' and 'b' having narrow and wide time windows, respectively. We reduced some of them by selecting only a random subset of the customers and decreasing the number of vehicles in an appropriate way; in this case we give a subscript denoting the index of the reduced instance. The initial set of columns is provided by taking the routes of feasible solutions of a variable neighborhood search described in [39]. All algorithms have been implemented in C++ using GCC 4.1 and were executed on a single core of a 2.2 GHz AMD Opteron 2214 PC with 4 GB RAM. CPLEX in version 11.1 was used as LP solver. The ESPPRC subproblem is solved in four alternative ways: (i) by dynamic programming (DP), (ii) by dynamic programming with a forced early stop after 1000 generated columns (DP^S), (iii) by ILS and a subsequent application of DP (ILS+DP), and finally (iv) by GRASP and running DP afterwards (GRASP+DP). The metaheuristics' iteration limit is originally set to 1000 and extended to 10000 if no new columns have been generated so far (on a per-run basis). DP is applied after the metaheuristic if less than 100 new columns have been generated. In all experiments, column generation was performed until the LP relaxation of the set covering formulation has been solved to optimality, i.e., it has been proven by DP that no further columns with negative reduced costs exist.

Table 3.2 shows the instances, the upper bounds (UB) initially provided by the variable neighborhood search, the (exact) lower bounds (LB) obtained from column generation, the percentage gaps between them, i.e. %-gap $= (\mathrm{UB} - \mathrm{LB})/\mathrm{LB} \cdot 100\%$, the CPU-times of settings DP and DP^S, as well as the minimal and average times of settings ILS+DP and GRASP+DP over ten runs per instance. It can be observed that DP^S is faster than DP for instances with narrow time windows, whereas it is almost the opposite for instances with wide time windows. However, using one of the metaheuristic combinations ILS+DP or GRASP+DP is almost always fastest, especially for larger instances, when the speed of the heuristic column generation outweighs the probably higher quality columns of the DP algorithm. Among the two metaheuristic combinations, no obvious advantage is observable for either of them.

Table 3.2 Experimental results of column generation with different pricing strategies for the PVRPTW: CPU-times for exactly solving the LP relaxation of the set covering formulation.

Instance							DP	DP^S	ILS+DP		GRASP+DP	
No.	n	m	t	UB	LB	%-gap	t[s]	t[s]	min t[s]	avg. t[s]	min t[s]	avg. t[s]
1a	48	3	4	2909.02	2882.01	0.94	5.01	4.87	11.44	13.92	13.40	18.59
2a	96	6	4	5032.06	4993.48	0.77	72.27	53.91	48.35	55.83	45.31	57.09
3a	144	9	4	7138.65	6841.44	4.34	374.07	291.61	231.17	265.28	223.03	262.89
$4a_{r1}$	160	10	4	6929.84	6641.67	4.34	1240.96	1136.64	605.06	744.69	640.15	754.71
7a	72	5	6	6784.71	6641.39	2.16	24.78	17.10	16.98	20.77	19.58	23.82
$9a_{r1}$	96	7	6	8545.80	8035.09	6.36	146.88	134.55	88.89	104.17	96.60	103.75
$9a_{r2}$	120	8	6	8598.40	8140.15	5.63	898.90	693.44	446.49	545.09	475.98	521.66
8a	144	10	6	9721.25	9153.79	6.20	745.95	592.07	367.82	418.87	383.18	421.63
$2b_{r1}$	32	2	4	2709.15	2682.52	1.00	89.65	121.30	43.58	64.79	25.36	55.78
1b	48	3	4	2277.44	2258.85	0.82	156.77	158.66	76.66	109.17	99.59	123.35
$2b_{r2}$	64	4	4	2771.68	2733.55	1.40	277.76	254.38	131.72	192.18	168.75	188.61
$3b_{r1}$	72	4	4	3306.86	3241.90	2.00	726.37	749.11	426.10	533.68	417.05	488.95
$7b_{r1}$	24	2	6	3776.25	3677.21	2.70	0.54	0.55	1.92	2.29	1.72	2.25
$8b_{r1}$	36	2	6	3640.79	3476.43	4.73	10.01	10.15	8.50	13.26	10.24	12.55
$7b_{r2}$	48	3	6	3723.18	3599.72	3.43	48.04	31.78	30.95	43.13	34.03	43.20
$8b_{r2}$	60	3	6	4606.17	4324.87	6.50	1538.18	1196.51	826.08	1121.28	804.35	967.87

3.8 Conclusions

The present chapter reviewed important MetaBoosting literature and presented two exemplary case studies where an in-depth description of successful hybrid algorithms was given. Many different hybridization approaches exist, most of them are specialized methods for specific problems, but another significant part of the surveyed research considers generic problems such as mixed integer programming. It is often possible to accelerate exact methods by introducing (meta-)heuristic knowledge and, if fixed time-limits are given, the overall solution quality might also benefit from such ideas.

In most exact approaches tight bounds are crucial aspects of success. Thus, different ways of applying metaheuristics for finding primal bounds were examined. The obvious way of determining high quality initial solutions can be beneficial to the overall optimization process, as it has been described for the multidimensional knapsack problem. General methods for determining improved primal solutions throughout the search process for generic mixed integer programming, such as local branching and relaxation or distance induced neighborhood search, have been found so effective that some of them have been included into the commercial MIP solver CPLEX. Solution merging approaches based on evolutionary algorithms were also successfully included in this solver. Other more problem specific methods often yielding optimal or close to optimal primal solutions based on the hybridization of Lagrangian relaxation and metaheuristics were also examined.

A multitude of collaborative approaches exist and some intertwined and parallel combinations were described in this chapter. Parallel combinations gain importance because of the current hardware developments and the broad availability of multi-core processors.

Mathematical programming techniques based on problem decomposition, such as cut and column generation approaches, play an essential role in the advances of exact methods. Applications where metaheuristic algorithms are used for such tasks were described. Especially sequential combinations of fast and simple construction heuristics and more sophisticated metaheuristic approaches are very promising in both cut and column generation.

Different aspects and difficulties in the development of hybrid methods were discussed in more detail in the two case studies. The first one describes a Lagrangian decomposition approach combined with an evolutionary algorithm for solving the knapsack constrained maximum spanning tree problem. The Lagrangian approach combined with an implicit construction heuristic and a subsequent local search is already a powerful procedure yielding tight gaps, but its combination with the EA allows to optimally solve substantially more of the large-scale instances. The second case study presents a column generation approach for solving the periodic vehicle routing problem with time windows. A greedy randomized adaptive search procedure, an iterated local search, and a dynamic programming algorithm are applied for solving the pricing subproblem. The inclusion of metaheuristic techniques led to a

significant acceleration of the column generation process compared to using the dynamic programming subproblem solver alone.

Hybridizing exact algorithms and metaheuristics, and MetaBoosting in particular are promising research areas. Further exciting results can be expected since various possible synergies are still unexplored. Especially generating, exchanging, and translating information about the ongoing optimization process by exploiting advanced features of the different algorithms will possibly lead to further progress in the field.

Acknowledgements This work is supported by the Austrian Science Fund (FWF) under contract number P20342-N13.

References

1. P. Augerat, J.M. Belenguer, E. Benavent, A. Corberan, and D. Naddef. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, 106(2):546–557, 1999.
2. E. Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13(4):517–549, 1965.
3. E. Balas and C.H. Martin. Pivot and complement – a heuristic for 0–1 programming. *Management Science*, 26(1):86–96, 1980.
4. F. Barahona and R. Anbil. The volume algorithm: Producing primal solutions with a subgradient method. *Mathematical Programming, Series A*, 87(3):385–399, 2000.
5. L. Bertaccoa, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4:63–76, 2007.
6. D. Bertsimas and J.N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.
7. A. Chabrier. Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, 33(10):2972–2990, 2006.
8. J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52:928–936, 2001.
9. E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming, Series A*, 102:71–90, 2005.
10. G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson. Solution of a large scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.
11. G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.
12. J. Denzinger and T. Offermann. On cooperation between evolutionary algorithms and other search paradigms. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 3, pages 2317–2324. IEEE Press, 1999.
13. O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1-3):229–237, 1999.
14. D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.
15. T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

16. G. Ribeiro Filho and L.A. Nogueira Lorena. Constructive genetic algorithm and column generation: an application to graph coloring. In L.P. Chuen, editor, *Proceedings of the Fifth Conference of the Association of Asian-Pacific Operations Research Societies within IFORS*, 2000.
17. M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.
18. M. Fischetti, C. Polo, and M. Scantamburlo. Local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. *Networks*, 44(2):61–72, 2004.
19. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming, Series B*, 98:23–47, 2003.
20. M.L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.
21. A. Frangioni. About Lagrangian methods in integer optimization. *Annals of Operations Research*, 139(1):163–193, 2005.
22. A.P. French, A.C. Robinson, and J.M. Wilson. Using a hybrid genetic algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. *Journal of Heuristics*, 7:551–564, 2001.
23. S. Ghosh. DINS, a MIP improvement heuristic. In M. Fischetti and D.P. Williamson, editors, *Integer Programming and Combinatorial Optimization: 12th International IPCO Conference, Proceedings*, volume 4513 of *Lecture Notes in Computer Science*, pages 310–323. Springer, 2007.
24. P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.
25. F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684, 2000.
26. F. Glover. Surrogate constraints. *Operations Research*, 16(4):741–749, 1968.
27. P. Hansen, N. Mladenović, and D. Urosević. Variable neighborhood search and local branching. *Computers & Operations Research*, 33(10):3034–3045, 2006.
28. M. Haouari and J.C. Siala. A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem. *Computers & Operations Research*, 33(5):1274–1288, 2006.
29. S.T. Henn. Weight-constrained minimal spanning tree problem. Master's thesis, University of Kaiserslautern, Department of Mathematics, May 2007.
30. F.S. Hillier. Efficient heuristic procedures for integer linear programming with an interior. *Operations Research*, 17(4):600–637, 1969.
31. S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors, *Column Generation*, chapter 2, pages 33–65. Springer, 2005.
32. J. Larsen. *Parallelization of the Vehicle Routing Problem with Time Windows*. PhD thesis, Technical University of Denmark, 1999.
33. M. Leitner and G.R. Raidl. Lagrangian decomposition, metaheuristics, and hybrid approaches for the design of the last mile in fiber optic networks. In M.J. Blesa, C. Blum, C. Cotta, A.J. Fernández, J.E. Gallardo, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics 2008*, volume 5296 of *Lecture Notes in Computer Science*, pages 158–174. Springer, 2008.
34. H.R. Lourenco, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, 2003.
35. M.E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.
36. S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0–1 knapsack problem. *Management Science*, 45:414–424, 1999.
37. G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

38. C. Oliva, P. Michelon, and C. Artigues. Constraint and linear programming: Using reduced costs for solving the zero/one multiple knapsack problem. In *International Conference on Constraint Programming, Proceedings of the Workshop on Cooperative Solvers in Constraint Programming*, pages 87–98, Paphos, Greece, 2001.
39. S. Pirkwieser and G.R. Raidl. A variable neighborhood search for the periodic vehicle routing problem with time windows. In C. Prodhon, R. Wolfler-Calvo, N. Labadi, and C. Prins, editors, *Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing*, Troyes, France, 2008.
40. S. Pirkwieser, G.R. Raidl, and J. Puchinger. A Lagrangian decomposition/evolutionary algorithm hybrid for the knapsack constrained maximum spanning tree problem. In C. Cotta and J. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 69–85. Springer, 2008.
41. J. Puchinger and G.R. Raidl. An evolutionary algorithm for column generation in integer programming: an effective approach for 2D bin packing. In In X. Yao, E.K. Burke, J.A. Lozano, J. Smith, J.J. Merelo-Guervos, J.A. Bullinaria, J.E. Rowe, P. Tino, A. Kaban, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 642–651. Springer, 2004.
42. J. Puchinger and G.R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In J. Mira and J.R. Álvarez, editors, *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, Part II*, volume 3562 of *Lecture Notes in Computer Science*, pages 41–53. Springer, 2005.
43. J. Puchinger and G.R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 183:1304–1327, 2007.
44. J. Puchinger, G.R. Raidl, and M. Gruber. Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem. In *Proceedings of the 6th Metaheuristics International Conference*, pages 775–780, Vienna, Austria, 2005.
45. G.R. Raidl and B.A. Julstrom. Edge sets: an effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, 7(3):225–239, 2003.
46. W. Rei, J.-F. Cordeau, M. Gendreau, and P. Soriano. Accelerating Benders decomposition by local branching. *INFORMS Journal on Computing*, in press.
47. E. Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.
48. M.W.P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4:146–154, 1992.
49. S. Talukdar, L. Baeretzen, A. Gove, and P. de Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4:295–321, 1998.
50. M. Vasquez and J.-K. Hao. A hybrid approach for the 0–1 multidimensional knapsack problem. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 328–333. Morgan Kaufman, 2001.
51. M. Vasquez and Y. Vimont. Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1):70–81, 2005.
52. Y. Vimont, S. Boussier, and M. Vasquez. Reduced costs propagation in an efficient implicit enumeration for the 0–1 multidimensional knapsack problem. *Journal of Combinatorial Optimization*, 15(2):165–178, 2008.
53. L.A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.
54. T. Yamada, K. Watanabe, and S. Katakoa. Algorithms to solve the knapsack constrained maximum spanning tree problem. *International Journal of Computer Mathematics*, 82(1):23–34, 2005.

Chapter 4
Usage of Exact Algorithms to Enhance Stochastic Local Search Algorithms

Irina Dumitrescu and Thomas Stützle

Abstract Exact mathematical programming techniques such as branch-and-bound or dynamic programming and stochastic local search techniques have traditionally been seen as being two general but distinct approaches for the effective solution of combinatorial optimization problems, each having particular advantages and disadvantages. In several research efforts true hybrid algorithms, which exploit ideas from both fields, have been proposed. In this chapter we review some of the main ideas of several such combinations and illustrate them with examples from the literature. Our focus here is on algorithms that have the main framework given by the local search and use exact algorithms to solve subproblems.

4.1 Introduction

Many problems arising in areas such as scheduling and production planning, location and distribution management, Internet routing or bioinformatics are combinatorial optimization problems (COPs). COPs are intriguing because they are often easy to state but often very difficult to solve, which is captured by the fact that many of them are $\mathcal{NP}$-hard [48]. This difficulty and, at the same time, their enormous practical importance, have led to a large number of solution techniques for them. The available solution techniques can be classified as being either *exact* or *approximate* algorithms. Exact algorithms are guaranteed to find an optimal solution and *prove* its optimality for every

Irina Dumitrescu
School of Mathematics, University of New South Wales, Sydney, Australia
e-mail: irina.dumitrescu@unsw.edu.au

Thomas Stützle
IRIDIA, Université Libre de Bruxelles (ULB), Brussels, Belgium
e-mail: stuetzle@ulb.ac.be

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_4,

finite size instance of a COP within an instance-dependent, finite run-time, or prove that no feasible solution exists. If optimal solutions cannot be computed efficiently in practice, it is usual to trade the guarantee of optimality for efficiency. In other words, the guarantee of finding optimal solutions is sacrificed for the sake of getting very good solutions in reasonably short time by using approximate algorithms.

Two solution method classes that have significant success are *integer programming* (IP) methods as an exact approach, and *stochastic local search* (SLS) algorithms [61] as an approximate approach. IP methods rely on the characteristic of the decision variables being integers. Some well known IP methods are branch-and-bound, branch-and-cut, branch-and-price, and dynamic programming [86]. In recent years, remarkable improvements have been reported for IP methods when applied to some problems (see, e.g. [11] for the TSP). Important advantages of exact methods for IP are that (i) *proven* optimal solutions can be obtained if the algorithm succeeds, (ii) valuable information on upper/lower bounds on the optimal solution are obtained even if the algorithm is stopped before completion (IP methods can become approximate if we define a criterion of stopping them before solving the problem), and (iii) IP methods allow to provably prune parts of the search space in which optimal solutions cannot be located. A more practical advantage of IP methods is that research code such as Minto [85] or GLPK [54], or powerful, general-purpose commercial tools such as CPLEX [63] or Xpress-MP [113] are available. However, despite the known successes, exact methods have a number of disadvantages. Firstly, for many problems the size of the instances that are practically solvable is rather limited and, even if an application is feasible, the variance of the computation times is typically very large when applied to different instances of a same size. Secondly, the memory consumption of exact algorithms can be very large and lead to the early abortion of a program. Thirdly, for many COPs the best performing algorithms are problem specific and they require large development times by experts in integer programming. Finally, high performing exact algorithms for one problem are often difficult to extend if some details of the problem formulation change. The state-of-the art for exact algorithms is that for some $\mathcal{NP}$-hard problems very large instances can be solved fast, while for other problems even small size instances are out of reach.

SLS is probably the most successful class of approximate algorithms. When applied to hard COPs, local search yields high-quality solutions by iteratively applying small modifications (local moves) to a solution in the hope of finding a better one. Embedded into higher-level guidance mechanisms, which are called (general-purpose) SLS methods [61]or, more commonly, metaheuristics, this approach has been shown to be very successful in achieving near-optimal (and often optimal) solutions to a number of difficult problems [1, 61, 108]. Examples of well-known general-purpose SLS methods (or metaheuristics) are simulated annealing, tabu search, memetic algorithms, ant colony optimization or iterated local search [52]. Advantages of SLS algorithms are

that (i) they are the best performing algorithms available for a variety of problems, (ii) they can examine a huge number of possible solutions in short computation time, (iii) they are often more easily adapted to slight variants of problems and are therefore more flexible, and (iv) they are typically easier to understand and implement by the common user than exact methods. However, local search based algorithms have several disadvantages. Firstly, they cannot prove optimality and typically do not give bounds on the quality of the solutions they return. Secondly, they typically cannot *provably* reduce the search space. Thirdly, they do not have well defined stopping criteria (this is particularly true for metaheuristics). Finally, local search methods often have problems with highly constrained problems where feasible areas of the solution space are disconnected. Another problem that occurs in practice is that, with very few exceptions [110], there are no efficient general-purpose local search solvers available. Hence, although one can typically develop an SLS algorithms of reasonable performance rather quickly, many applications of SLS algorithms can require considerable development and implementation efforts if very high performance is required.

It should be clear by now that IP and SLS approaches have their particular advantages and disadvantages and can be seen as complementary. Therefore, it appears to be a good idea to try to combine these two distinct techniques into more powerful algorithms. Many articles use the probably most straightforward of these combinations, which is the use of an SLS algorithm to compute good intial upper bounds (in the minimization case) that allow an early pruning of non-optimal solutions. Intuitively, one would expect more advanced combinations of exact and local search methods to result in even stronger algorithms.

In this chapter we focus on approaches that strive for an integration of IP and SLS techniques. In particular, we describe ways of integrating IP methods (or methods derived from an IP approach) into SLS methods. In other words, we focus on approaches where the main algorithm (the master) is the local search technique and the IP method is typically used to solve some subproblems. For an overview of methods where as the master method can be considered the IP method and SLS techniques are used to solve subproblems, we refer to Chapter 3 of this book. We refer to this chapter and Chapter 2 for methods that use the spirit of local search inside IP methods (such as local branching [46]).

More in detail, we will cover the following five ideas that can be abstracted from the various true hybrid algorithms proposed in the literature: (i) the usage of exact algorithms to explore large neighborhoods in SLS algorithms, (ii) ways of enhancing metaheuristics by solving some subproblems exactly, (iii) the usage of techniques taken from branch-and-bound to enhance constructive SLS methods, (iv) the exploitation of the structure of good solutions identified by SLS algorithms, and (v) the exploitation of information from relaxations in SLS algorithms. In fact, the first three ways of combining local search and exact algorithms are classified in [96] as integrative combinations,

while the last two are classified as collaborative combinations: the two methods are executed in sequence. For each of these five groups, we will describe some general ideas, give one or more detailed examples, and give pointers to literature for other ideas following the corresponding idea.

Before proceeding with our discussion we need to clarify the notion of *solution component* and *decision variable* that we will use in our chapter. We explain what we mean by using the example of the symmetric traveling salesman problem (TSP). The TSP can be defined on an undirected graph $G = (V, E)$, where V is the set of nodes and E the set of edges. An edge represents a pair of nodes between which travel is possible. Every edge $\{i, j\}$ has an associated cost d_{ij} and the goal in the TSP is to find a Hamiltonian circuit of minimal length. When applying local search to the TSP, a solution component usually refers to a single edge in a tour; each tour contains exactly n such solution components. More generally, a solution component can be seen as an atomic item and a set of such atomic items defines a problem solution. The analogue to a solution component in IP formulations is the notion of decision variable. For example, in the usual IP formulations of the TSP (see [86] for an overview), a variable $x_{ij} \in \{0, 1\}$ is associated with each edge $\{i, j\}$. The variable x_{ij} is equal to 1, if the edge $\{i, j\}$ is included in a tour and zero, otherwise. The x_{ij} variables are called decision variables. In the following sections, if we say that a decision variable is fixed to one (zero), we mean that we consider only those solutions in which this variable is set to one (zero). Analogously, in the local search case we say that a solution component is forced to always occur (not occur) in any solution considered. In the case of the TSP, this corresponds to an edge always (never) being used in any of the solutions considered. If a variable is free, it can take any value in $\{0, 1\}$.

4.2 Exploring large neighborhoods

Given a current solution s, (perturbative) local search [61] explores the neighborhood $\mathcal{N}(s)$ of s iteratively and tries to replace s by a solution $s' \in \mathcal{N}(s)$ according to some criterion. In local search, it is appealing to search large neighborhoods because much better solutions can be reached in one local search step than when using simple, small neighborhoods. However, large neighborhoods have the associated disadvantage that a considerable amount of time may be spent to search them in order to find some or the best improving neighboring solution. More than that, many of the large neighborhoods proposed in the literature are exponentially large with respect to instance size [51, 70, 93, 106, 107] and the main potential bottleneck for local search in large neighborhoods is the task of searching for a better or the best solution in the neighborhood of the current one.

Local search algorithms for large neighborhoods can be classified roughly into two classes. The first class comprises those where the large neighborhood is searched heuristically in some problem-specific way. Examples of this class of local search algorithms are variable-depth search algorithms [70, 65] or ejection chains [51]. The second are local search algorithms where the neighborhood is searched exactly, i.e. the central idea of these algorithms is to model the problem of searching a large neighborhood as an optimization problem, which is solved exactly. The solution of the resulting search problem will determine the neighbor that will replace the current solution in the local search.

Two main possibilities are considered for defining the neighborhood search problem. The first one is to define a special-purpose neighborhood and to model the exploration of the *full* neighborhood as an optimization problem. In this case, a search problem is defined such that each feasible solution of it induces a move of the local search algorithm. Clearly, the task of the exact algorithm in this case will be to solve the resulting *neighborhood search problem* (NSP). A general algorithmic outline of this idea can be given as follows:

Algorithm 4.2.1 *Neighborhood search*

Step 1: *Initialization*
Let s be a feasible initial solution.
Step 2: *Local search*
while `stopping_criterion` is not met **do**
Define a search problem $\mathcal{P}(s)$ that depends on s.
Find $opt(\mathcal{P}(s))$, an optimal solution of $\mathcal{P}(s)$.
Let s' be the solution induced by $opt(\mathcal{P}(s))$.
if s' is better than s w.r.t. the objective function **then** $s = s'$.
enddo
Step 3: Return s.

The stopping criterion may be triggered, e.g., if no improved solution s can be obtained or the solutions to $\mathcal{P}(s)$ are infeasible etc. The first two examples that we present, very large scale neighborhood search and Dynasearch, fall in this class of NSP hybrids. Typically, algorithms that are based on NSP try to determine composite moves, which are composed of a sequence or set of simpler moves.

The second "possibility" is that at each step of the local search a part of the current solution s is maintained fixed, thus defining a partial solution, while the values of the rest of the decision variables are left free. The free part is then rearranged optimally, subject to the constraint that the fixed part cannot be altered. In this sense, the task of the exact algorithm is to solve the *partial neighborhood search problem* (PNSP). The following algorithmic outline gives a high level view of such a procedure in a first-improvement style of local search. The outline sketches an iterative improvement algorithm

that stops in the first locally optimal solution encountered. The operator $\otimes$, which is used below, joins two partial solutions into a complete solution. A distinctive feature to the NSP type algorithms is that the final move is not considered to be composed of simple underlying moves.

Algorithm 4.2.2 *Partial neighborhood search*

Step 1: *Initialization*
Let s be an initial solution.
Step 2: *Neighborhood search*
while `improvement found` **do**
Step 3: *Neighborhood scan*
while not full neighborhood examined **do**
Delete solution components from solution s resulting in a partial solution: $s_p = s \setminus r$.
Define a search problem $\mathcal{P}(r)$.
Find $opt(\mathcal{P}(r))$, the optimal solution of $\mathcal{P}(r)$.
Perform the move induced by $opt(\mathcal{P}(r))$, resulting in solution $\bar{s}'$; $s' = s_p \otimes \bar{s}'$.
if s' is better than s w.r.t. the objective function **then** $s = s'$.
enddo
enddo
Step 4: Return s.

We illustrate the partial neighborhood search principle on the hyperopt local search algorithm. In fact, hyperopt is only one example of such methods. They can be traced back to Applegate and Cook's shuffle heuristic [12].

4.2.1 NSP Example: Cyclic and Path Exchange Neighborhoods

One application of NSP is the very large scale neighborhood (VLSN) search that was introduced by Ahuja et al. [5, 6] in the context of solving partitioning problems. Examples of partitioning problems include graph coloring, vehicle routing, generalized assignment, parallel machine scheduling, clustering, aggregation, and graph partitioning. Next, we briefly describe the type of problems to which VLSN search is applied and how the neighborhood search problem is defined.

A partition $\mathcal{T}$ of a set W of n elements is a set of subsets $\mathcal{T} = \{T_1, \ldots, T_K\}$ such that $W = T_1 \cup \cdots \cup T_K$ and $T_k \cap T_{k'} = \emptyset$, $k, k' = 1, \ldots, K$. To each set T_k one can associate a cost $c_k(T_k)$. Then, the partitioning problem is to search for a partition of W such that the sum of the costs of the partitions is minimal. The properties of the cost function are not important for now, except that it is separable over subsets.

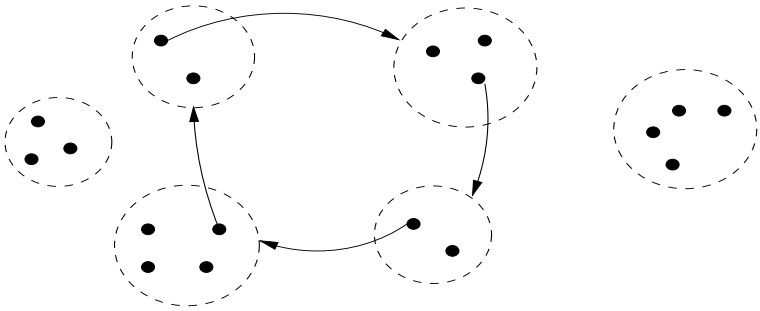


Fig. 4.1 Example of a cyclic exchange for a partitioning problem.

The one-exchange neighborhood, which is based on the move of one element of W from one partition into another one, and the two-exchange neighborhood, which is based on exchanges of two elements of two different partitions, are typical simple neighborhoods for partitioning problems. Cyclic exchange neighborhoods generalize the two-exchange neighborhood [6, 106, 107] by exchanging not only two elements from two different subsets but moving several elements, each of a different subset. An illustration of this neighborhood is given in Figure 4.1. A path exchange is similar to a cyclic exchange except that from the last subset of the path no element is removed.

The cyclic (path) exchange neighborhood of a partition $\mathcal{T}$ is then the set of all partitions $\mathcal{T}'$ that can be obtained by a cyclic (path) exchange as described above. In what follows we focus on the cyclic exchange neighborhood. A cyclic exchange modifies the cost of the partitions involved in the move. If the cost function is separable over subsets, the cost of a cyclic exchange can be obtained as the sum of the cost differences of all subsets that are modified by the move.

Ahuja et al. define an *improvement graph* on which a search problem is solved in order to find an improving cyclic exchange. The node set V of the improvement graph is defined as the collection of integers $1, \ldots, n$ such that each node in V is in one-to-one correspondence to an element of W. For each pair of elements that do not belong to the same subset in $\mathcal{T}$, an arc (i, j) between the corresponding nodes is added in the improvement graph if and only if the subset to which the element corresponding to node j belongs remains feasible after the removal of the element corresponding to j and the addition of the element corresponding to i. The partition $\mathcal{U}$ of V is the collection of subsets $U_1, \ldots, U_K$ that corresponds to the subsets in $\mathcal{T}$. Hence, we have that a cyclic exchange in $\mathcal{T}$ corresponds to a subset disjoint cycle in the improvement graph with respect to $\mathcal{U}$. The cost associated to any arc (i, j) is defined to be equal to the difference between the cost of the set after the removal of the element corresponding to j and the addition of the element corresponding to i, and the cost of the original set that contains the element corresponding to j. Thompson and Orlin [106] showed that there is

a one-to-one correspondence between the cyclic exchanges with respect to $\mathcal{T}$ and the subset-disjoint cycles in the improvement graph (with respect to $\mathcal{U}$) and that both have the same cost.

The search for a best neighboring solution can then be modeled as a search problem on the improvement graph. In fact, for determining an improving cyclic exchange move, a subset disjoint negative cost cycle (SDNCC) needs to be found in the improvement graph, i.e. a cycle that uses at most one node from every subset of the partition of the set of nodes. Determining the best among the improving neighbors corresponds to finding the most negative such cylce (this problem is called the SDNCCP) and needs to be solved exactly. Several exact methods for the SDNCCP are proposed by Dumitrescu [40]. She also proposes algorithms for the more general problem of determining a subset disjoint minimum cost cycle (SDMCC), which includes the case when such a minimum cost cycle need not be negative (i.e., it does not correspond to an improvement). Such cases are important if, e.g. the VLSN technique is embedded into a tabu search. The algorithms for determining the optimal solutions to SDNCCP and SDMCCP can be seen as generalizations of dynamic programming algorithms for shortest path problems. An acceleration method that exploits symmetries for the SDMCCP is given, and an elegant theorem of Lin and Kernighan [70] is exploited in the SDNCCP case. Although both problems are $\mathcal{NP}$-hard, the proposed algorithms have been shown to be very efficient for the subproblems that arise in practical applications of VLSN search methods. Dumitrescu also proposes heuristics that are obtained by limiting or truncating the exact methods in some way.

An exact solution of SDNCCP was also presented by Ahuja et al. [7]. They integrated the exact solution of the SDNCCP into a VLSN local search for the capacitated minimum spanning tree problem and obtained very good results, improving the best known solutions for 36% of the tested benchmark instances. A more recent study of an exact evaluation of cyclic and path exchange neighborhoods for graph coloring has been presented in [33]. Although the usage of these neighborhoods gave significant improvements with respect to the solution quality reachable for iterative improvement algorithms for graph coloring [33], once these neighborhoods have been integrated into metaheuristic algorithms, the overhead in computation time made the approach undesirable. Since this may happen also for other problems, it is not surprising that the exact solution of the neighborhood search problem in cyclic and path exchange neighborhoods is sometimes replaced by a heuristic solution to the SDNCCP or SDMCCP, which are, however, often derived from a possible exact solution of the neighborhood search problem.

4.2.2 NSP Example: Dynasearch

Dynasearch is a local search technique that uses a dynamic programming algorithm to search for the best possible combination of mutually independent, simple search moves to be executed in one local search iteration. In the context of dynasearch, the independence of the simple moves refers to the requirements that they do not interfere with each other with respect to (i) the evaluation function being used and (ii) the feasibility of a solution after the execution of the moves. In particular, the independence of the moves with respect to the evaluation function implies that the gain obtained by a dynasearch move can be obtained as the sum of the gains of the simple moves.

So far, various applications of dynasearch to permutation problems have been proposed. The independence between simple search moves is typically related to the fact that the indices affected by the simple moves are not overlapping. Let $\pi = (\pi(1), \ldots, \pi(n))$ be the current permutation. Two moves that involve elements from $\pi(i)$ to $\pi(j)$ and $\pi(k)$ to $\pi(l)$, with $1 \leq i < j \leq n$ and $1 \leq k < l \leq n$ need to have that either $j < k$ or $l < i$.

The best combination of independent moves can be obtained by a dynamic programming algorithm, which gives the name of this technique. Let $\Delta(j)$ be the maximum total cost reduction obtained from independent moves affecting only positions 1 to j of the current permutation and let $\delta(i, j)$ be the cost reduction by a move involving positions between i and j, including i and j. The idea of the algorithm is to compute the maximum total cost reduction obtained by either appending element $\pi(j)$ to the partial permutation of the elements $\pi(1)$ to $\pi(j-1)$ or by appending element $\pi(j)$ and applying a move involving element $\pi(j)$ (and possibly elements from $\pi(i)$ onwards).

The values of $\Delta(j), j = 0, \ldots, 1$ can be computed in a dynamic programming fashion. Let π be the current solution and set $\Delta(0) = 0$ and $\Delta(1) = 0$. Then, $\Delta(j),\ j = 1, \ldots, n-1$ can be obtained in a forward evaluation by the recursive formula

$$\Delta(j+1) = \max\{\max_{1 \leq i \leq j}\{\Delta(i-1) + \delta(i, j+1)\}, \Delta(j)\}. \tag{4.1}$$

$\Delta(n)$ gives the largest reduction in solution cost. The compound move to be executed can then be determined by tracing back the computation steps.

When applying the algorithm, the particularities of the problem and of the moves may have to be taken into account to slightly adapt the dynamic programming recursion given by Equation (4.1). (Note that the dynasearch algorithm can be cast in terms of a search for a shortest path in an appropriately defined improvement graph [3].)

Current applications of dynasearch comprise the TSP [34], the single machine total weighted tardiness problem (SMTWTP) [34, 35, 55] and a time-dependent variant of it [8], and the linear ordering problem (LOP) [34]. A general observation is that dynasearch, on average, is faster than a standard best-improvement descent algorithm and may return slightly better qual-

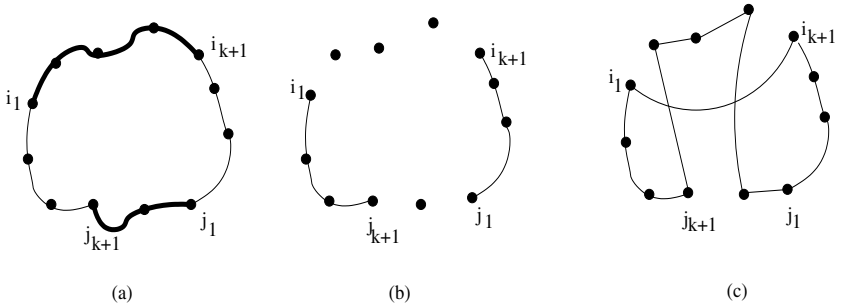


Fig. 4.2 (a) A feasible tour can be seen as the union of $\mathcal{H}(i_1, i_{k+1})$, $\mathcal{H}(i_{k+1}, j_1)$, $\mathcal{H}(j_1, j_{k+1})$, and $\mathcal{H}(j_{k+1}, i_1)$. (b) The hyperedges $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ are removed. (c) A new feasible tour is constructed.

ity solutions. Particularly good performance is reported when dynasearch is used as a local search routine inside other metaheuristics such as iterated local search [73]. Currently, iterated dynasearch is the state-of-the-art method for SMTWTP [55], and very good results were obtained for the TSP and the LOP [34].

4.2.3 PNSP Example: Hyperopt Neighborhoods

A relatively simple example of a PNSP approach is the hyperopt local search approach applied to the TSP [27, 37, 28]. To keep the presentation as simple as possible, we will only consider its application to the symmetric TSP.

The *hyperopt neighborhood* is based on the notion of hyperedges. Given a tour of the TSP, a *hyperedge* is defined to be a subpath of the tour; in other words, a sequence of successive edges of the tour [37]. If i is the start node and j the end node of the hyperedge, we denote the hyperedge by $\mathcal{H}(i, j)$. The *length* of a hyperedge is given by the number of edges in it. Let t be a feasible tour of the TSP and $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ two hyperedges of length k such that $\mathcal{H}(i_1, i_{k+1}) \cap \mathcal{H}(j_1, j_{k+1}) = \emptyset$ with respect to the nodes contained. We assume that the tour t can be written as $t = (i_1, \ldots, i_{k+1}, \ldots, j_1, \ldots, j_{k+1}, \ldots, i_1)$. It is obvious that the tour t can be described completely by four hyperedges: $\mathcal{H}(i_1, i_{k+1})$, $\mathcal{H}(i_{k+1}, j_1)$, $\mathcal{H}(j_1, j_{k+1})$, and $\mathcal{H}(j_{k+1}, i_1)$, as shown in Figure 4.2(a). A *k-hyperopt move* is defined as a composition of the two following steps: remove $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ from the tour t, then add edges to $\mathcal{H}(i_{k+1}, j_1)$ and $\mathcal{H}(j_{k+1}, i_1)$ such that a new feasible tour is constructed (see Figure 4.2 for an example). The *k-hyperopt neighborhood* consists of all k-hyperopt moves.

The size of the k-hyperopt neighborhood increases exponentially with k. The authors propose an "optimal" construction of a k-hyperopt move by

solving exactly a subproblem: the TSP defined on the graph $G' = (V', E')$, where V' is the set of nodes included in $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ and E' is the set of edges in the original TSP that have both ends in V'. However, this approach is bound to be efficient only when k is relatively small. Otherwise, a large size TSP would have to be solved as a subproblem. To avoid this, the authors propose to use a dynamic programming algorithm for the case $k \geq 3$ [28, 37], but no details of the algorithm are given.

For a given hyperedge $\mathcal{H}(i_1, i_{k+1})$ the k-hyperopt move to be performed is determined in [28, 37] in a first descent form. Hence, a sequence of subproblems is generated and solved, one subproblem for every hyperedge that does not intersect with $\mathcal{H}(i_1, i_{k+1})$. To speed up computations, usual TSP speed-up techniques such as "don't look bits" are used [19]. In addition, extensions based on embeddings of the hyperopt local search into iterated local search and variable neighborhood search algorithms are considered. However, the currently available results suggest that the resulting hyperopt local search for the ATSP is somewhat inferior to current state-of-the-art local search techniques [64]. In spite of this, the k-hyperopt approach may have potential if the size of the neighborhood is greatly enlarged and truly efficient algorithms such as Concorde [10] are used to solve the resulting subproblems.

4.2.4 Other Approaches

The two NSP techniques for exploring a neighborhood that we have described in this section have been applied to a variety of problems. For a recent overview of VLSN search applications we refer to the book chapter by Ahuja et al. [4]. Recently, the corridor method has been proposed by Sniedovich and Voß [103]. The central idea put forward is to define constraints on the problems, such that efficient exact methods can be designed to solve the neighborhood search problem efficiently. Another example of an NSP method is the constraint programming approach by Pesant and Gendreau [89, 90], where the exploration of the neighborhood is modeled as a problem that is then solved with constraint programming techniques [78, 109]. Constraint programming is an especially promising approach, if the resulting search problems are tightly constrained.

PNSP algorithms have been more widely applied. As mentioned before, one of the first PNSP algorithms is the shuffle heuristic of Applegate and Cook [12], which was applied to the job-shop scheduling problem. Briefly, their approach consists of defining a partial solution by leaving the sequence fixed for one or a few machines. The subproblem of completing the schedule is then solved by branch-and-bound. Another example is the MIMAUSA algorithm, introduced by Mautor and Michelon [79, 80, 81], applied to the quadratic assignment problem. In MIMAUSA, at each step k variables are "freed". The subproblem of assigning values to the free variables, such that

the rest of the solution is kept fixed, is solved to optimality. A similar approach has been applied by Büdenbender et al. [26] to the direct flight network design problem. Hu et al. [62] search large neighborhoods for the generalized minimum spanning tree problem. In particular, they implemented a variable neighborhood descent algorithm, where in the largest neighborhood a mixed-integer program (MIP) solver is used to determine optimal solutions to the subproblems arising in the local search. Computational results show that their approach obtained excellent performance. A similar approach of searching large neighborhoods by a MIP solver is applied by Prandtstetter and Raidl to a car sequencing problem [94]. PNSP methods have been successfully applied in combination with constraint programming techniques. Some noteworthy examples are the work of Caseau and Laburthe [31] for the job-shop scheduling problem or that of Shaw [102] for the vehicle routing problem, where a branch-and-bound algorithm is integrated into tabu search. An earlier overview of combinations of local search and constraint programming is given in an article by Focacci et al. [47].

4.2.5 Discussion

In general, the use of exact algorithms for exploring large neighborhoods is appealing. For some problems the resulting neighborhoods can be explored fully by polynomial time algorithms, a noteworthy example being here the dynasearch approach. If the large neighborhoods cannot be explored in *provably* polynomial time, often the resulting neighborhood search problems, whether corresponding to full or partial neighborhood, can be solved efficiently by exact algorithms. This is mainly due to the exact algorithms being, in most cases, rather quick if the problem size is not too large and to the fact that the resulting subproblems often have a special structure that can be exploited efficiently.

Despite the usefulness of exact algorithms in these methods, it is clear that the resulting subproblems could be solved by approximate algorithms. In fact, this is often done. A general framework for this idea of defining subproblems and exploring them heuristically, POPMUSIC, was defined in a paper of Taillard and Voß [104]. Similar ideas were put forward in variable neighborhood decomposition search [58]. It would be interesting to study more carefully the use of heuristics versus the use of exact algorithms for exploring the large neighborhoods using sound experimental designs and try to derive more general insights into when one is prefered of the other.

4.3 Enhancing Metaheuristics

Occasionally, exact methods are employed to implement central sub-procedures in a metaheuristic, typically with the goal of adding specific features of diversification or intensification of the search that are beyond the reach of local search algorithms. One early example of such a combination is the usage of exact algorithms as perturbation operators in iterated local search.

4.3.1 Example: Perturbation in Iterated Local Search

A successful approach to solving difficult COPs is iterated local search (ILS) [73]. ILS consists of running a local search procedure, starting from solutions obtained by perturbing available local optima. A simple form of an ILS is given in Algorithm 4.3.1.

Algorithm 4.3.1 *Iterated Local Search*

Step 1: Let s be an initial solution.
Step 2: **while** `stopping_criterion` is not met **do**
(i) Let s' be the solution obtained from s after a perturbation.
(ii) Call `local_search`(s') to produce the solution s''.
(iii) **if** s'' is accepted as the new incumbent solution **then** $s = s''$.
enddo
Step 3: Return s.

The main role of the perturbation in ILS is to introduce significant changes in the current solution in order to allow the local search to explore different local optima, while still conserving good characteristics of a current solution. Simple ways of introducing such perturbations, like applying a random move in a large neighborhood, are usually enough to obtain good performance. However, state-of-the-art performance is often reached by more problem specific perturbations and when the perturbations introduce some structural changes that cannot be reversed easily by a local search [73].

One possibility of combining exact algorithms with ILS is to let an exact algorithm determine the perturbation. This may be very useful, since the exact algorithm may introduce larger structural changes, which cannot be easily undone by the local search. The implementation of this idea can be done by first fixing some part of the solution and leaving the rest free. Next, the free part is optimized, and the solution to the free part is reinserted into the fixed part, possibly after restoring feasibility if this should be necessary. In other words, at Step 2(i) of Algorithm 4.3.1 a subproblem is solved to optimality. Formally, Step 2(i) is replaced by:

Algorithm 4.3.2 *Solving a subproblem to determine the perturbation (new Step 2(i) of ILS)*

begin

Let $s'' = s \setminus r$, where $r \subset s$.
Define $\mathcal{P}(r)$, a subproblem that depends on r.
Let $opt(\mathcal{P}(r))$ be the optimal solution of $\mathcal{P}(r)$.
Let $\bar{s'} = s'' \cup opt(\mathcal{P}(r))$.
Modify $\bar{s'}$ such that feasibility is restored.
Return s', be the modified feasible solution.

end

An early example of how to determine a perturbation by solving a subproblem exactly is given by Lourenço [72]. In her paper, Lourenço conducts an extensive computational study of ILS applied to the job-shop scheduling problem (JSP). The JSP is defined for m machines and n jobs. Each job consists of a sequence of operations that have to be performed in a given order. Each operation has to be executed for a specified, uninterrupted time (i.e. preemption is not allowed). In the JSP, precedence constraints induce a total order on the operations of each job. Additional constraints require that each machine handles at most one job at a time. A feasible schedule is a schedule that satisfies all the precedence and capacity constraints. The JSP consists in finding a feasible schedule that minimizes the overall job completion time.

The JSP can be modeled using the disjunctive graph $G = (V, A, E)$ representation, where V is the vertex set corresponding to the operations, A is the arc set corresponding to the job precedence constraints, and E is the edge set corresponding to the machine capacity constraints [101]. In this graph, the scheduling decision corresponds to orienting each edge in E.

Lourenço experimentally tested several ways of defining the perturbation. One perturbation procedure proposed by Lourenço is making use of Carlier's algorithm [30], which is a branch-and-bound method applied to a one-machine scheduling problem. The particular problem solved here can be seen as a very simple version of the JSP: a number of operations need to be scheduled on one machine in the presence of temporal constraints. Lourenço's idea is to modify the directed graph corresponding to the current solution of the JSP by removing all the directions given to the edges associated with two randomly chosen machines. Then Carlier's algorithm is applied to one of the machines. The problem to solve is therefore a one-machine scheduling problem. Next, the edges corresponding to that machine are oriented according to the optimal solution obtained. Finally, the same treatment is applied to the second machine. Lourenço mentions that this perturbation idea can create cycles in the graph and suggests a way of obtaining a feasible schedule from the graph with cycles (see [72] for details). In conclusion, at each iteration of the ILS, two subproblems are solved in order to construct a new initial solution for the local search procedure. Lourenço proposed similar perturbation schemes for her ILS algorithm for the JSP; for details we refer to [71, 72].

Finally, it should be said that the computational results of Lourenço cannot be considered state-of-the-art performance. The main reason probably is that she used a rather weak local search algorithm when compared to the

effective tabu search algorithms proposed by Nowicki and Smutnicki [87] or the local search procedure of Balas and Vazacopoulos [15]. However, it would be interesting to see how the performance of these two algorithms would be affected by the addition of the perturbation steps proposed by Lourenço.

4.3.2 Other Approaches

Exact algorithms can be used within particular metaheuristics for specific operations that are to be done while searching for solutions. An example in a somewhat similar spirit as the previously described one is followed in the recent paper by Fernandes and Lourenço [43]. They introduce occasional large perturbations into a tabu search algorithm, which consists in a partial destruction of a solution and a reconstruction of a new one. The reconstruction is guided by using violated valid inequalities that enforce some specific order among unscheduled operations. Apart from the perturbations in ILS, other examples can be found in applications of genetic algorithms. Here, exact algorithms are typically used in the recombination operation, in which two solutions s and s' are combined to form one or several new solutions (also called offspring). The main idea is to define a subproblem `Recombination`(s, s') that comprises all the solutions that can result following a particular way of combining s and s', and then to search for the best solution of the resulting subproblem.

The subproblem can be obtained by keeping common solution features of the two "parent" solutions fixed and to try to find the best solution for the free parts. A second possibility is to define a subproblem consisting of the union of the solution components contained in the two parent solutions s and s'. An example for the first approach is presented by Yagiura and Ibaraki [114]. They apply this idea to permutation problems using a dynamic programming algorithm for finding an optimal permutation subject to the constraint that a partial order common to the two parent solutions s and s' is maintained by the new solution. Other examples include the MIP-recombination operator for a specific supply management problem [24] and for the balancing transfer lines problem described in Chapter 7. There, many of the variables that have the same value in the parents are fixed and the resulting subproblem is solved by a mixed-integer programming solver. The complexity of optimal recombination when solutions are represented as bitstrings is studied in [42]. Examples of the second idea are the papers of Balas and Niehaus [14] on the maximum clique problem (MCP), and of Aggarwal et al. [2] on the maximum independent set problem (MISP). The MCP and MISP are subset problems in graphs, where a subset of the vertices needs to be chosen. Hence, in both cases, a solution corresponds to a subset of the vertices of the original graph $G = (V, A)$, i.e. we have $s, s' \subset V$. In both cases, the subproblem consists of a subgraph comprising all the vertices in $s \cup s'$ and all edges that connect

them. It can be shown that the resulting subproblem is a matching problem in a bipartite graph. Hence, it can be solved in polynomial time [2, 13]. Another paper following these lines is that of Lourenço et al. [74] on a multi-objective bus driver scheduling problem, where analogous ideas are applied to the offspring determination of a genetic algorithm applied to a set covering problem.

4.3.3 Discussion

Clearly, the techniques discussed here are specific to particular metaheuristics. However, these examples indicate possible areas of interest, where such combinations can be useful. In general, mainly hybrid metaheuristics will be candidates for possible combinations. We call hybrid metaheuristics those metaheuristics that consist of the combination of several clearly distinct procedures like perturbation and local search in the ILS case or recombination, mutation, and (possibly) local search in the genetic algorithm case. In fact, the examples outlined above illustrate well the potential of such combinations. Other metaheuristics that could profit from such combinations are ant colony optimization [38], e.g. by optimizing partial solutions or extending partial solutions in an optimal way, or scatter search [53], in a way analogous to what is described for the genetic algorithms.

4.4 Using Branch-and-Bound Techniques in Constructive Search Heuristics

Construction heuristics build solutions starting from an empty partial solution. They iteratively add solution components until a complete solution is obtained. Construction heuristics typically rate the desirability of adding a solution component based on a heuristic measure of its objective function contribution. Then, they either add a solution component greedily, i.e. by choosing the best rated component, or probabilistically, biased by the heuristic information.

In construction heuristics partial solutions are typically only extended and never reduced. However, construction heuristics can easily by transformed into tree search algorithms, e.g. by adding a backtracking-type mechanism and usages of lower and upper bounds. This finally leads to branch-and-bound algorithms [61].Hence, a somewhat natural hybrid is to extend construction heuristics or metaheuristics that are based on the repeated construction of solutions such as ant colony optimization [39] by tree search techniques.

4.4.1 Example: Approximate Nondeterministic Tree Search (ANTS)

Ant Colony Optimization (ACO) [39] is a recent metaheuristic approach for solving hard COPs. In ACO, (artificial) ants are stochastic solution construction procedures that probabilistically build a solution by iteratively adding solution components to partial solutions while taking into account (i) heuristic information on the problem instance being solved, if available, and (ii) (artificial) pheromone trails. Pheromone trails in ACO serve as distributed, numerical information which is adapted during the execution of the algorithm to reflect the search experience. Of the available ACO algorithms, the approximate nondeterministic tree search (ANTS) algorithm [75] was the first to integrate branch-and-bound techniques into ACO.

ANTS was first applied to the quadratic assignment problem (QAP). In the QAP, a set of objects has to be assigned to a set of locations with given distances between the locations and given flows between the objects. The goal in the QAP is to place the objects on locations in such a way that the sum of the product between flows and distances is minimal. More formally, in the QAP one is given n objects and n locations, values a_{ij} representing the distance between locations i and j, and b_{rs} representing the flow between objects r and s. Let x_{ij} be a binary variable which takes value 1 if object i is assigned to location j, and 0 otherwise. The problem can be formulated as follows:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n} a_{ij}b_{kl}x_{ik}x_{jl} \tag{4.2}$$

subject to the constraints

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n \tag{4.3}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,\ldots,n \tag{4.4}$$

$$x_{ij} \in \{0,1\} \qquad i,j = 1,\ldots,n \tag{4.5}$$

In ANTS, each ant constructs a solution by iteratively assigning objects to a free location. Hence, the solution components to be considered are pairings between objects and locations (there are n^2 such solution components). Given a location j, an ant decides to assign object i to this location with a probability given by

$$p_{ij}^{k}(t) = \frac{\alpha \cdot \tau_{ij}(t) + (1-\alpha)\cdot \eta_{ij}}{\sum_{l\in\mathcal{N}_j^k} \alpha \cdot \tau_{lj}(t) + (1-\alpha)\cdot \eta_{lj}} \qquad \text{if } i \in \mathcal{N}_j. \tag{4.6}$$

Here, $\tau_{ij}(t)$ is the pheromone trail associated to the assignment of object i to a location j, which gives the "learned" desirability of choosing this assignment (pheromones vary at run-time), η_{ij} is the heuristic desirability of this assignment, α is a weighting factor between pheromone and heuristic, and $\mathcal{N}_j$ is the feasible neighborhood, i.e. the set of objects that are not yet assigned to a location.

Lower bound computations are exploited at various places in ANTS. Before starting the solution process, ANTS first computes the Gilmore-Lawler lower bound (GLB) [49, 68]. The GLB is defined to be the solution to the assignment problem min $\sum_{i=1}^{n} \sum_{j=1} c_{ij}x_{ij}$ subject to constraints (4.3) to (4.5). The coefficients c_{ij} are defined as a function of distances and flows and the variables x_{ij} are binary. It is known that the assignment problem has the integrality property. Therefore a solution of it can be obtained by simply solving its linear programming relaxation. Along with the lower bound computation one gets the values of the dual variables u_i and v_i, $i = 1, \ldots, n$, corresponding to the constraints (4.3) and (4.4), respectively. The dual variables v_i are used to define a pre-ordering on the locations: the higher the value of the dual variable associated to a location, the higher the impact of the location on the QAP solution cost is assumed to be. Hence, the assignment of an object to that location is tried earlier.

The essential idea of ANTS is to use computations on the lower bounds on the completion cost of a partial solution in order to define the heuristic information about the attractiveness of adding a specific solution component (i, j) in (4.6). This is achieved by tentatively adding the solution component to the current partial solution and by estimating the cost of a complete solution, (containing that solution component), by means of a lower bound. This estimate is then used to influence the probabilistic decisions taken by the ant during the solution construction: the lower the estimate, the more attractive the addition of a specific pair. A further advantage of the usage of a lower bound is that an extension of a partial solution, for which the lower bound estimate is higher than the cost of the best solution found so far, can be discarded.

A disadvantage of the GLB, which is computed in an initialization phase of the algorithm, is that its computation is $\mathcal{O}(n^3)$. This is clearly expensive, since a lower bound has to be computed at each step during a solution construction of an ant. Therefore, Maniezzo proposes a lower bound weaker than GLB, the so-called LBD bound, which exploits the values of the dual variables to estimate the completion cost of a partial solution. Although LBD can be shown to be weaker than GLB, the main advantage of using LBD is its low computational complexity, which is $\mathcal{O}(n)$. For details on the lower bound computation we refer to [75]. Experimental results have shown that ANTS was at the time it was proposed one of the best available algorithms for the QAP. The good performance of the ANTS algorithm has been confirmed in a variety of further applications [77, 76].

4.4.2 Other Approaches

The integration of tree search techniques into constructive algorithms is an appealing possibility of hybridization. Given that ACO is, apart from GRASP, probably the most widespread used constructive metaheuristic, it is not astonishing that a number of other hybrids between ACO and tree search algorithms have been proposed. Probably the most noteworthy is the beam-ACO algorithm, which combines ACO with beam search, a derivative of branch-and-bound algorithms. beam search is a tree search algorithm that keeps at each iteration a set of nodes of a search tree and expands each of them in several directions according to a selection based on lower bounds [88]. beam-ACO takes from ACO the probabilistic extension of partial solutions and uses a beam search type of management of partial solutions. It has shown very high performance for problems such as open shop scheduling [20] and assembly line balancing [21]. More details on beam-ACO and similar hybrid algorithms can be found in [22]. A combination of a GRASP algorithm with a branch-and-bound algorithm is described in [44] for an application to the JSP.

Another active area of hybridization is the integration of constraint programming techniques into constructive algorithms and, again, in particular into ACO. Such approaches have been described first by Meyer and Ernst [84] and in some more detail in a book chapter [83]. More recently, the integration of ACO into constraint programming languages has been considered [66].

4.5 Exploiting the Structure of Good Solutions

Many optimization problems show some type of structure in the sense that high quality solutions have a large number of solution components in common with optimal solutions. This observation is exploited by some hybrid methods. These define appropriate subproblems of the original problem by using the information obtained from high quality solutions. Often the resulting subproblems are small enough to be solved rather efficiently by exact algorithms.

This approach consists of two phases, which are executed in sequence. In the first phase, an approximate algorithm is used repeatedly to collect a number of high-quality solutions of the problem under consideration. Based on the solution components that are included in the set of solutions, a subproblem of the original problem is defined. It is expected that the subproblem still contains all or, if not all, most of the "important" decision variables and that the subproblem can be solved relatively easily by an exact algorithm. The optimal solution for the subproblem provides an upper bound for the optimal solution of the original problem.

An algorithmic outline of the type of methods falling into this framework is given next.

Algorithm 4.5.1 *Exploiting structure by collecting information*

Step 1: *Initialization*
Let $\mathcal{I} = \emptyset$, where $\mathcal{I}$ is the set of collected solutions.
Step 2: *Approximate algorithm*
while `stopping_criterion` is not met **do**
Let s be the solution returned after running an approximate algorithm.
Add s to $\mathcal{I}$.
enddo
Reduce $\mathcal{I}$ according to some criteria (optional).
Step 3: *Optimization*
Define a subproblem $\mathcal{P}(\mathcal{I})$ that depends on $\mathcal{I}$.
Find $opt(\mathcal{P}(\mathcal{I}))$, the optimal solution of $\mathcal{P}(\mathcal{I})$.
Step 4: Return $opt(\mathcal{P}(\mathcal{I}))$.

Here we discuss two well known examples where this idea was successfully applied to the p-median problem and the TSP.

4.5.1 Example: Heuristic Concentration

Rosing and ReVelle designed the heuristic concentration algorithm for the solution of the p-median problem [99]. Given a graph $G = (V, A)$, where $V = \{1, \ldots, n\}$, a weight associated with each node, and a distance associated with each arc, and given a positive integer p, the p-median problem consists of finding p nodes in the graph, called *facilities*, such that the sum of the weighted distances between every node and its closest facility is minimized. The nodes that are not facilities are called *demand nodes*.

The technique developed by Rosing and ReVelle is called *heuristic concentration*. In the first phase, a local search procedure, in this case the Teitz and Bart heuristic [105], is run repeatedly with different random starts. Every solution obtained in this process is recorded in a set $\mathcal{I}$. The number of times the heuristic is run is determined after conducting numerical experiments.

The second phase starts with the selection of a subset $\mathcal{I}'$ of the solutions in $\mathcal{I}$. $\mathcal{I}'$ contains only the best solutions of $\mathcal{I}$ with respect to the objective function. The number of solutions in $\mathcal{I}'$ is again determined by numerical experiments. The facility locations used in these "best" solutions form a subset of the set of all nodes. They will be collected in what the authors call the *concentration set* (CS). Finally, a restricted p-median problem, with the facility locations restricted to those contained in the concentration set, is solved.

Clearly, this method is built on the assumption that a near-optimal or optimal solution must have the facilities located at sites from CS. However, this is only an assumption based on experimental results. An even stronger assumption is made by the authors in order to further reduce the dimension of the subproblem being solved. This is the claim that the nodes that are facilities in *all* solutions considered *must* be facilities. The rest of the CS contains *possible* locations. Therefore, they split CS into two subsets: one contains the locations where there are facilities in all solutions, called *CS open* (CS_0), and the other one contains the nodes where there is a facility in at least one but not all solutions. The latter subset of CS is called *CS free* (CS_f). The authors exploit the fact that each demand node not in CS must be assigned to the closest facility. For every node, variables are needed only for that node in CS_0 that is closest to the demand node, and only for those nodes in CS_f that are even closer than that member of the CS_0.

It is clear that when CS can be split, the size of the second subproblem is smaller than the size of the first one. However, there is no guarantee that such a partition of CS exists. When $CS_0 = \emptyset$, the only possibility is to use the first subproblem. Since in most cases the solution of the LP relaxation of the binary integer model of a p-median problem is integer, Rosing and ReVelle solve the LP relaxation of the restricted problem using an LP solver (in this case CPLEX). Even when the solution is fractional, an integer solution is found very easily. Numerical results are provided in both [99] and subsequent papers [98, 100]. They prove that the heuristic concentration is a viable technique that obtains very good results in practice.

4.5.2 Example: Tour Merging

Applegate et al. [9] proposed the tour merging approach, which was further refined by Cook and Seymour [36]. Tour merging is a two phase procedure following the steps outlined above.

The first phase consists of generating a number of high quality tours for the TSP instance that needs to be solved. While Applegate et al. [9] used their chained Lin-Kernighan (LK) algorithm, in the later study in [36] the iterated version of Helsgaun's LK implementation [60] is used. This heuristic reaches significantly better quality solutions and using it to define the set of tours $\mathcal{I}$ in the first stage was found to result in better quality tours in the second stage. Additionally, given that for the same number of tours the number of decision variables is smaller, larger instances can be tackled in this way.

The second phase consists of solving the TSP on the graph induced by the set of tours $\mathcal{I}$ on the original graph. In other words, a TSP is solved on a restricted graph that has the same set of nodes as the original graph, but its set of edges consists of the edges that appear at least once in any of the tours

in $\mathcal{I}$. Formally, if the original graph is $G = (V, A)$, where V is the set of nodes and A the set of arcs, the reduced graph can be described as $G' = (V, A')$, where $A' = \{a \in A : \exists t \in \mathcal{I}, a \in t\}$.

Several possibilities of solving the TSP on the reduced graph G' can be considered. In [9], an exact solver part of the Concorde package, was used. When Concorde was applied to medium sized instances (with up to 5000 cities), the tour merging method could identify optimal solutions for all instances at least twice out of ten independent trials. However, on few instances the computation time was large, with most of it being taken by the optimization code applied in the second stage. Another possibility of solving the TSP on G' is to use special purpose algorithms that exploit the structure of the graph G'. In particular, G' is a very sparse graph and it is likely to have a low *branch-width* [97]. In [36], a dynamic programming algorithm that exploits this property is presented and computational results on a wide range of large TSP instances (2,000 up to about 20,000 cities) show excellent performance, resulting in computation times for the second stage that are much shorter than the times required by a general purpose TSP solver. In fact, the tour merging approach is currently one of the best performing approximate algorithms for medium to large TSP instances.

4.5.3 Discussion

The two examples show that approaches based on collected information can result in highly efficient hybrid algorithms in practice. For the approach to work on specific problems, high-quality solutions need to have specific characteristics. Firstly, it is important that these solutions have many components in common. Then the resulting subproblem will have a small number of decision variables and will be amenable to solving with exact algorithms. Secondly, the resulting subproblem is required to contain most of (ideally, all) the important decision variables to ensure that the second stage can return a very high quality solution. Such properties appear to hold in the TSP case: it is a well known property that local minima cluster in a small region of the search space and that the better their quality, the more edges they have in common with an optimal solution [23, 67, 82]. Some evidence exists that similar conclusions hold for the p-median problem [57]. Notions of search space analysis are important in this context. Intuitively, problems with a high *fitness distance correlation* are more likely to be amenable for such algorithmic approaches. Roughly speaking, a high fitness distance correlation indicates that the better the solutions the closer they are to an optimal solution in terms of an appropriate distance definition.

Clearly, the ideas outlined in this section are not necessarily restricted to using an exact algorithm in the second stage. For example, in the case of the set covering problem (SCP), Finger et al. [45] first provide a fitness

distance correlation analysis of the SCP search space. Based on the result that for some classes of SCP instances the characteristics mentioned above were satisfied, a subproblem is defined by the solutions returned by several local searches and then solved by simulated annealing [25]. Puchinger et al. [95] study the "core" concept [16, 92] for the multidimensional knapsack problem and apply SLS algorithms (a memetic algorithm and a relaxation guided variable neighborhood search) and exact algorithms to the resulting reduced instances. They obtain very high quality solutions and show that the SLS algorithms reach better quality solutions on the "core" instances than when applying them directly to the original instances. (Note that the concept of "cores" is often defined through linear relaxations, e.g. by fixing variables that are integer in the relaxation; these approaches are related to those that are discussed in Section 4.6.)

Finally, it should be mentioned that the approaches described in this section are related to some already discussed in Section 4.3. The "optimized crossover" algorithms described there define a new problem similar to the union of solution components in tour merging or heuristic concentration. In fact, the latter two could be seen as an "optimized multi-parent recombination" operator. A difference is that the approaches presented here were originally conceived as consisting of two stages that are executed in sequence. However, alternative and intermediate approaches can easily be conceived.

4.6 Exploiting Information from Relaxations in Metaheuristics

A more frequent way of combining elements from exact algorithms and metaheuristics is the exploitation of some type of information obtained from a relaxation of the original problem to bias in some way the SLS algorithm. Various possibilities have been explored from rather straightforward uses to more complex ones. From the latter ones we present an example given by Vasquez and Hao [111] for the multidimensional knapsack problem. We then give some pointers to more literature on the exploitation of relaxations.

4.6.1 Example: Simplex and Tabu Search Hybrid

Vasquez and Hao [111] combine a tabu search with an exact method, in this case a simplex algorithm that solves some linear programming subproblems. Given m resources, each having a corresponding resource limit, n objects, each with an associated profit, and an associated vector of resource consumption, the 0-1 multidimensional knapsack problem seeks to maximize the total profit made by using the objects such that the amount of resources con-

sumed is within the resource limits. In what follows we will denote the profit associated with object i by p_i, the amount of resource j consumed by selecting object i by r_{ji}, and the limit of resource j by R_j. The standard integer programming formulation of the 0-1 multidimensional knapsack problem can be written as:

$$\max \sum_{i=1}^{n} p_i x_i$$

$$\text{s.t.} \sum_{i=1}^{n} r_{ji} x_i \leq R_j, \quad \forall j = 1, \ldots, m \tag{4.7}$$

$$x_i \in \{0, 1\}, \qquad \forall i = 1, \ldots, n, \tag{4.8}$$

where $x_i = 1$ if object i is used and $x_i = 0$ otherwise.

The method starts by relaxing the integrality constraint. Since the solution of the linear programming relaxation of an integer programming problem can be far away from the integer optimal solution, the linear programming relaxation is then "strengthened" by the addition of an extra constraint. It is clear that the integer optimal solution of the 0-1 multidimensional knapsack problem can have only a certain number of components that are *not* zero. Based on this observation, the constraint added to the linear programming relaxation enforces the number of non-zero components to be equal to k, where $0 \leq k \leq n$ (see constraint (4.9) below). Therefore $n + 1$ linear programming problems are obtained (one for each value of k):

$$\max \sum_{i=1}^{n} p_i x_i$$

$$\text{s.t.} \sum_{i=1}^{n} r_{ji} x_i \leq R_j, \quad \forall j = 1, \ldots, m$$

$$\sum_{i=1}^{n} x_i = k \tag{4.9}$$

$$x_i \in [0, 1], \qquad \forall i = 1, \ldots, n \tag{4.10}$$

We denote such a subproblem by $\mathcal{P}(k)$. Clearly, the solutions of these problems can be fractional. However, it is hoped that the optimal solution of the original problem is close to one of the optimal solutions obtained after solving the relaxed problems. (To reduce the number of subproblems that need to be solved, Vasquez and Hao compute bounds on k; for details see [111].)

After the reduction, each linear program is solved by the simplex algorithm. The solutions returned are then used to generate starting solutions to a following tabu search phase. More precisely, an initial solution is obtained from a possibly fractional solution $x^{[k]}$ to $\mathcal{P}(k)$ by fixing its largest k elements to one and the rest to zero. The solution $x^{[k]}$ of $\mathcal{P}(k)$ has the additional use

to restrict the search space explored by a tabu search procedure. In fact, the subsequently run tabu search algorithm, which is based on the reverse elimination method [50], is restricted to explore only solutions x for which it holds that $\sum_{i=1}^{n} |x_i - x_i^{[k]}| \leq \delta_{max}$, i.e. the solutions explored by the tabu search are limited to a ball of radius δ_{max} around $x^{[k]}$. The radius δ_{max} is computed heuristically as follows. Let no_int denote the number of integer components of $x^{[k]}$ and no_frac the number of fractional components; then, the radius is set to a value of $\delta_{max} \approx 2 \times (no\_int + no\_frac - k)$. Note that $\delta_{max} = 0$ corresponds to the case when an integer solution is returned by simplex. The search space is further reduced by limiting the number of objects taken into configurations. For each k, the number of non-zero components of a candidate solution is considered to be exactly k and only the *integer* candidate solutions are considered. This actually corresponds to a partitioning of the search space. The neighborhoods are defined as add/drop neighborhoods. The neighborhood of a given configuration is the set of configurations that differ by exactly two elements from the initial configuration, while keeping the number of non-zero elements constant.

An extensive computational study of the final algorithm is provided. The authors report many improved results compared to the best known results at the time of publication; however, they acknowledge large computational time of about three days on a set of computers with Pentium 300MHz or 500 MHz computers for very large instances with as many as 2,500 items and 100 constraints. Further improvements on this approach are described in [112].

4.6.2 Discussion

The paper of Hao and Vasquez is an example of how information of solutions of relaxations of IP formulations can be used to restrict the area of the search space examined by local search and to provide good initial solutions for the local search. As said before, there are a number of other approaches that in some form make use of relaxations and information derived from them. How relaxations and problem decompositions can be used to define heuristic methods and metaheuristics is shown in Chapter 5 of this book.

A particularly active area appears to be the exploitation of lower bound computations through Lagrangean relaxation (LR). (For an introduction to Lagrangean relaxation and its use in combinatorial optimization we refer to [18].) Such approaches have been proposed for a variety of problems and for several they result in state-of-the-art algorithms. The central idea in these approaches is to use Lagrangean multipliers and costs to guide construction or local search type heuristics. Among the first approaches that use LR is the heuristic of Beasley [17]. Currently, state-of-the-art algorithms for the set covering problem are, at least in part, based on exploiting information from LR [29, 32]. Ways of integrating information gained in LR into a tabu

search algorithm is given in [56] and computational results are presented for an example application to the capacitated warehouse location problem. Other, recent examples of using LR to direct SLS algorithms include a genetic algorithm for the prize collecting Steiner tree problem [59], exploiting Lagrangean decomposition for an evolutionary algorithm for the knapsack constrained maximum spanning tree problem [91], or hybrid approaches to the design of the last mile in fiber optic networks [69].

4.7 Conclusions

In this chapter we reviewed existing approaches that combine stochastic local search and methods from the area of integer programming for the solution of $\mathcal{NP}$-hard combinatorial optimization problems, extending significantly over our earlier review [41]. We focused on techniques where the "master" routine is based on some stochastic local search method and strengthened by the use of exact algorithms; see Chapter 3 for the case where the IP method is the master. In the terms used in [96], the discussed approaches can be classified as integrative or as collaborative, sequential approaches. In the former class (covered here in Sections 4.2 – 4.4), the exact algorithms are used to repeatedly solve subproblems that arise in the SLS algorithm. In the latter type of approaches (discussed in Sections 4.5 and 4.6), an exact methods is executed either before the start of the SLS part, typically, to produce information that is later used to bias the local search, or used as a post-processing facility to improve solutions generated by typically several local search processes.

The main conclusion we can make here is that there are many research opportunities to develop algorithms that integrate local search and exact techniques. Despite the fact that local search and exact algorithms have somewhat complementary advantages and disadvantages, relatively few researches present algorithms that try to combine the two areas. One reason for this may be that such combined methods are often rather complex and hence they may involve significantly long development times. A possibly more important obstacle is that they require strong knowledge about the techniques available in two rather different techniques, which are often treated in different research streams. Fortunately, this situation appears to be improving, as exemplified by the contributions in this book. In general, we strongly believe that combinations of exact methods and stochastic local search methods are a very promising direction for future research in combinatorial optimization.

Acknowledgements This work has been supported by META-X, an ARC project funded by the French Community of Belgium. TS acknowledges support from the fund for scientific research F.R.S.-FNRS of the French Community of Belgium, of which he is a research associate.

References

1. E.H.L. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, 1997.
2. C.C. Aggarwal, J.B. Orlin, and R.P. Tai. An optimized crossover for the maximum independent set. *Operations Research*, 45:226–234, 1997.
3. R.K. Ahuja, Ö.Ergun, J.B. Orlin, and A.P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1–3):75–102, 2002.
4. R.K. Ahuja, Ö. Ergun, J.B. Orlin, and A.P. Punnen. Very large-scale neighborhood search: Theory, algorithms, and applications. In T.F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, pages 20–1—20–15. Chapman & Hall/CRC, Boca Raton, FL, 2007.
5. R.K. Ahuja, J.B. Orlin, and D. Sharma. Multi-exchange neighbourhood structures for the capacitated minimum spaning tree problem. Working Paper, 2000.
6. R.K. Ahuja, J.B. Orlin, and D. Sharma. Very large-scale neighbourhood search. *International Transactions in Operational Research*, 7(4–5):301–317, 2000.
7. R.K. Ahuja, J.B. Orlin, and D. Sharma. A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. *Operations Research Letters*, 31(3):185–194, 2003.
8. E. Angel and E. Bampis. A multi-start dynasearch algorithm for the time dependent single-machine total weighted tardiness scheduling problem. *European Journal of Operational Research*, 162(1):281–289, 2005.
9. D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding Tours in the TSP. Technical Report 99885, Forschungsinstitut für Diskrete Mathematik, University of Bonn, Germany, 1999.
10. D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. Concorde TSP solver. `http://www.tsp.gatech.edu//concorde`, last visited December 2008.
11. D. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ, 2006.
12. D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.
13. E. Balas and W. Niehaus. Finding large cliques in arbitrary graphs by bipartite matching. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26, pages 29–53. American Mathematical Society, 1996.
14. E. Balas and W. Niehaus. Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. *Journal of Heuristics*, 4(2):107–122, 1998.
15. E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2):262–275, 1998.
16. E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28(5):1130–1154, 1980.
17. J.E. Beasley. A Lagrangian heuristic for set covering problems. *Naval Research Logistics*, 37:151–164, 1990.
18. J.E. Beasley. Lagrangean relaxation. In C.R. Reeves, editor, *Modern heuristic techniques for combinatorial problems*, pages 243–303. Blackwell Scientific Publications, 1993.
19. J.L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.
20. C. Blum. Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591, 2005.

21. C. Blum. Beam-ACO for simple assembly line balancing. *INFORMS Journal on Computing*, 20(4):618–627, 2008.
22. C. Blum, C. Cotta, A.J. Fernández, J.E. Gallardo, and M. Mastrolilli. Hybridizations of metaheuristics with branch & bound derivatives. In C. Blum, M.J. Blesa, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics—An Emergent Approach to Optimization*, volume 117 of *Studies in Computational Intelligence*, pages 85–116. Springer Verlag, Berlin, 2008.
23. K.D. Boese, A.B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimization. *Operations Research Letters*, 16:101–113, 1994.
24. P. Borisovsky, A. Dolgui, and A. Eremeev. Genetic algorithms for a supply management problem: MIP-recombination vs. greedy decoder. *European Journal of Operational Research*, 195(3):770–779, 2009.
25. M.J. Brusco, L.W. Jacobs, and G.M. Thompson. A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated set covering problems. *Annals of Operations Research*, 86:611–627, 1999.
26. K. Büdenbender, T. Grünert, and H.-J. Sebastian. A hybrid tabu search/branch-and-bound algorithm for the direct flight network design problem. *Transportation Science*, 34(4):364–380, 2000.
27. E.K. Burke, P. Cowling, and R. Keuthen. Embedded local search and variable neighbourhood search heuristics applied to the travelling salesman problem. Technical report, University of Nottingham, 2000.
28. E.K. Burke, P.I. Cowling, and R. Keuthen. Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem. In E.J.W. Boers, J. Gottlieb, P.L. Lanzi, R.E. Smith, S. Cagnoni, E. Hart, G.R. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computing*, volume 2037 of *Lecture Notes in Computer Science*, pages 203–212. Springer Verlag, Berlin, 2001.
29. A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations Research*, 47:730–743, 1999.
30. J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.
31. Y. Caseau and F. Laburthe. Disjunctive scheduling with task intervals. Technical Report LIENS 95-25, Ecole Normale Superieure Paris, France, July 1995.
32. S. Ceria, P. Nobili, and A. Sassano. A Lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81(2):215–228, 1995.
33. M. Chiarandini, I. Dumitrescu, and T. Stützle. Very large-scale neighborhood search: Overview and case studies on coloring problems. In C. Blum, M.J. Blesa, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics—An Emergent Approach to Optimization*, volume 117 of *Studies in Computational Intelligence*, pages 117–150. Springer Verlag, Berlin, 2008.
34. R.K. Congram. *Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimization*. PhD thesis, Southampton University, Faculty of Mathematical Studies, Southampton, UK, 2000.
35. R.K. Congram, C.N. Potts, and S.L. Van de Velde. An iterated dynasearch algorithm for the single–machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.
36. W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.
37. P.I. Cowling and R. Keuthen. Embedded local search approaches for routing optimization. *Computers & Operations Research*, 32(3):465–490, 2005.
38. M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. In Glover and Kochenberger [52], pages 251–285.
39. M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.
40. I. Dumitrescu. *Constrained Shortest Path and Cycle Problems*. PhD thesis, The University of Melbourne, 2002.

41. I. Dumitrescu and T. Stützle. Combinations of local search and exact algorithms. In G.R. Raidl, J.A. Meyer, M. Middendorf, S. Cagnoni, J.J.R. Cardalda, D.W. Corne, J. Gottlieb, A. Guillot, E. Hart, C.G. Johnson, and E. Marchiori, editors, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 211–223. Springer Verlag, Berlin, 2003.
42. A.V. Eremeev. On complexity of optimal recombination for binary representations of solutions. *Evolutionary Computation*, 16(1):127–147, 2008.
43. S. Fernandes and H.R. Lourenço. Optimised search heuristic combining valid inequalities and tabu search. In M.J. Blesa, C. Blum, C. Cotta, A.J. Fernández, J.E. Gallardo, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics, 5th International Workshop, HM 2008*, volume 5296 of *Lecture Notes in Computer Science*, pages 87–101. Springer Verlag, Berlin, 2008.
44. S. Fernandes and H.R. Lourençou. A simple optimised search heuristic for the job shop scheduling problem. In C. Cotta and J.I. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 203–218. Springer Verlag, Berlin, 2008.
45. M. Finger, T. Stützle, and H.R. Lourençou. Exploiting fitness distance correlation of set covering problems. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G.R. Raidl, editors, *Applications of Evolutionary Computing*, volume 2279 of *Lecture Notes in Computer Science*, pages 61–71. Springer Verlag, Berlin, 2002.
46. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming, Series B*, 98:23–47, 2003.
47. F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming. In Glover and Kochenberger [52], pages 369–403.
48. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, CA, 1979.
49. P.C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the SIAM*, 10:305–313, 1962.
50. F. Glover. Tabu search. *ORSA Journal on Computing*, 2:4–32, 1990.
51. F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65:223–253, 1996.
52. F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, MA, 2002.
53. F. Glover, M. Laguna, and R. Martí. Scatter search and path relinking: Advances and applications. In Glover and Kochenberger [52], pages 1–35.
54. GLPK (GNU Linear Programming Kit). `http://www.gnu.org/software/glpk/glpk.html`, last visited December 2008.
55. A. Grosso, F. Della Croce, and R. Tadei. An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Operations Research Letters*, 32(1):68–72, 2004.
56. T. Grünert. Lagrangean tabu search. In P. Hansen and C.C. Ribeiro, editors, *Essays and Surveys on Metaheuristics*, pages 379–397. Kluwer Academic Publishers, Boston, MA, 2002.
57. P. Hansen and N. Mladenoviç. Variable neighbourhood search for the p-median. *Location Science*, 5(4):207–226, 1998.
58. P. Hansen, N. Mladenovic, and D. Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.
59. M. Haouari and J.C. Siala. A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem. *Computers & Operations Research*, 33(5):1274–1288, 2006.
60. K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.
61. H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, 2005.

62. B. Hu, M. Leitner, and G.R. Raidl. Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *Journal of Heuristics*, pages 473–499, 2008.
63. ILOG. `http://www.ilog.com/products/cplex/`, 2008.
64. D.S. Johnson and L.A. McGeoch. Experimental analysis of heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, 2002.
65. B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technology Journal*, 49:213–219, 1970.
66. M. Khichane, P. Albert, and C. Solnon. Integration of ACO in a constraint programming language. In M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, and A.F.T. Winfield, editors, *Ant Colony Optimization and Swarm Intelligence, 6th International Conference, ANTS 2008*, volume 5217 of *Lecture Notes in Computer Science*, pages 84–95. Springer Verlag, Berlin, 2008.
67. S. Kirkpatrick and G. Toulouse. Configuration space analysis of travelling salesman problems. *Journal de Physique*, 46(8):1277–1292, 1985.
68. E.L. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963.
69. M. Leitner and G.R. Raidl. Lagrangian decomposition, metaheuristics, and hybrid approaches for the design of the last mile in fiber optic networks. In M.J. Blesa, C. Blum, C. Cotta, A.J. Fernández, J.E. Gallardo, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics, 5th International Workshop, HM 2008*, volume 5296 of *Lecture Notes in Computer Science*, pages 158–174. Springer Verlag, Berlin, 2008.
70. S. Lin and B.W. Kernighan. An effective heuristic algorithm for the travelling salesman problem. *Operations Research*, 21:498–516, 1973.
71. H.R. Lourenço. *A Computational Study of the Job-Shop and the Flow-Shop Scheduling Problems*. PhD thesis, School of Or & IE, Cornell University, Ithaca, NY, 1993.
72. H.R. Lourenço. Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83:347–367, 1995.
73. H.R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In Glover and Kochenberger [52], pages 321–353.
74. H.R. Lourenço, J.P. Paixão, and R. Portugal. Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation Science*, 35(3):331–343, 2001.
75. V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.
76. V. Maniezzo and A. Carbonaro. An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16(8):927–935, 2000.
77. V. Maniezzo, A. Carbonaro, M. Golfarelli, and S. Rizzi. An ANTS algorithm for optimizing the materialization of fragmented views in data warehouses: Preliminary results. In *Applications of Evolutionary Computing, EvoWorkshops 2001*, volume 2037 of *Lecture Notes in Computer Science*, pages 80–89. Springer Verlag, Berlin, 2001.
78. K. Marriott and P. Stuckey. *Programming with Constraints*. MIT Press, Cambridge, MA, 1998.
79. T. Mautor. Intensification neighbourhoods for local search methods. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 493–508. Kluwer Academic Publishers, Norwell, MA, 2002.
80. T. Mautor and P. Michelon. MIMAUSA: A new hybrid method combining exact solution and local search. In *Extended abstracts of the 2nd International Conference on Meta-heuristics*, page 15, Sophia-Antipolis, France, 1997.
81. T. Mautor and P. Michelon. MIMAUSA: an application of referent domain optimization. Technical Report 260, Laboratoire d'Informatique, Université d'Avignon et des Pays de Vaucluse, Avignon, France, 2001.

82. P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 244–260. McGraw Hill, London, UK, 1999.
83. B. Meyer. Hybrids of constructive metaheuristics and constraint programming. In C. Blum, M.J. Blesa, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics—An Emergent Approach to Optimization*, volume 117 of *Studies in Computational Intelligence*, pages 85–116. Springer Verlag, Berlin, 2008.
84. B. Meyer and A. Ernst. Integrating ACO and constraint propagation. In M. Dorigo, M. Birattari, C. Blum, L.M. Gambardella, F. Mondada, and T. Stützle, editors, *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004*, volume 3172 of *Lecture Notes in Computer Science*, pages 166–177. Springer Verlag, Berlin, 2004.
85. MINTO - Mixed INTeger Optimizer. `http://coral.ie.lehigh.edu/minto`, last visited December 2008.
86. G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.
87. E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job-shop problem. *Management Science*, 42(2):797–813, 1996.
88. P.S. Ow and T.E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:297–307, 1988.
89. G. Pesant and M. Gendreau. A view of local search in constraint programming. In E. Freuder, editor, *Proceedings of Constraint Programming 1996*, volume 1118 of *Lecture Notes in Computer Science*, pages 353–366. Springer Verlag, Berlin, 1996.
90. G. Pesant and M. Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5:255–279, 1999.
91. S. Pirkwieser, G.R. Raidl, and J. Puchinger. A Lagrangian decomposition / evolutionary algorithm hybrid for the knapsack constrained maximum spanning tree problem. In C. Cotta and J.I. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 69–85. Springer Verlag, Berlin, 2008.
92. D. Pisinger. Core problems in knapsack algorithms. *Operations Research*, 47(4):570–575, 1999.
93. C.N. Potts and S. van de Velde. Dynasearch: Iterative local improvement by dynamic programming; part I, the traveling salesman problem. Technical Report LPOM–9511, Faculty of Mechanical Engineering, University of Twente, Enschede, The Netherlands, 1995.
94. M. Prandtstetter and G.R. Raidl. An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. *European Journal of Operational Research*, 191(3):1004–1022, 2008.
95. J. Puchinger, G.R. Raidl, and U. Pferschy. The core concept for the multidimensional knapsack problem. In J. Gottlieb and G.R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization–EvoCOP 2006*, volume 3906 of *Lecture Notes in Computer Science*, pages 195–208. Springer Verlag, Berlin, 2006.
96. G.R. Raidl and J. Puchinger. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In C. Blum, M.J. Blesa, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics—An Emergent Approach to Optimization*, volume 117 of *Studies in Computational Intelligence*, pages 31–62. Springer Verlag, Berlin, 2008.
97. N. Robertson and P.D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory*, 52:153–190, 1991.
98. K.E. Rosing. Heuristic concentration: a study of stage one. *Environment and Planning B: Planning and Design*, 27(1):137–150, 2000.
99. K.E. Rosing and C.S. ReVelle. Heuristic concentration: Two stage solution construction. *European Journal of Operational Research*, pages 955–961, 1997.

100. K.E. Rosing and C.S. ReVelle. Heuristic concentration and tabu search: A head to head comparison. *European Journal of Operational Research*, 117(3):522–532, 1998.
101. B. Roy and B. Sussmann. Les problemes d'ordonnancement avec constraintes disjonctives. Notes DS no. 9 bis, SEMA.
102. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming - CP98, 4th International Conference*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer Verlag, Berlin, 1998.
103. M. Sniedovich and S. Voß. The corridor method: A dynamic programming inspired metaheuristic. *Control and Cybernetics*, 35:551–578, 2006.
104. É.D. Taillard and S. Voß. POPMUSIC: Partial optimization metaheuristic under special intensification conditions. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in metaheuristics*, pages 613–629. Kluwer Academic Publishers, Boston, MA, 2002.
105. M.B. Teitz and P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16:955–961, 1968.
106. P.M. Thompson and J.B. Orlin. The theory of cycle transfers. Working Paper No. OR 200-89, 1989.
107. P.M. Thompson and H.N. Psaraftis. Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.
108. P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.
109. P. van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, Cambridge, MA, 1999.
110. P. Van Hentenryck and L. Michel. *Constraint-Based Local Search*. MIT Press, Cambridge, MA, 2005.
111. M. Vasquez and J.-K. Hao. A hybrid approach for the 0-1 multidimensional knapsack problem. In B. Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 328–333. Morgan Kaufmann Publishers, San Francisco, CA, 2001.
112. M. Vasquez and Y. Vimont. Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1):70–81, 2005.
113. Xpress-MP. `http://www.dashoptimization.com/home//products/products_optimizer.html`, last visited December 2008.
114. M. Yagiura and T. Ibaraki. The use of dynamic programming in genetic algorithms for permutation problems. *European Journal of Operational Research*, 92:387–401, 1996.

Chapter 5
Decomposition Techniques as Metaheuristic Frameworks

Marco Boschetti, Vittorio Maniezzo, and Matteo Roffilli

Abstract Decomposition techniques are well-known as a means for obtaining tight lower bounds for combinatorial optimization problems, and thus as a component for solution methods. Moreover a long-established research literature uses them for defining problem-specific heuristics. More recently it has been observed that they can be the basis also for designing metaheuristics. This tutorial elaborates this last point, showing how the three main decomposition techniques, namely Dantzig-Wolfe, Lagrangean and Benders decompositions, can be turned into model-based, dual-aware metaheuristics. A well known combinatorial optimization problem, the Single Source Capacitated Facility Location Problem, is then chosen for validation, and the implemented codes of the proposed algorithms are benchmarked on standard instances from literature.

5.1 Introduction

Traditionally, heuristic methods, and metaheuristics in particular, have been primal-only methods. They are usually quite effective in solving the given problem instances, and they terminate providing the best feasible solution found during the allotted computation time. However, disregarding dual information implies some obvious drawbacks, first of all not knowing the quality of the proposed solution, but also having possibly found an optimal solution at the beginning of the search and having wasted CPU time ever since, having

Marco Boschetti
Department of Mathematics, University of Bologna, Bologna, Italy
e-mail: marco.boschetti@unibo.it

Vittorio Maniezzo · Matteo Roffilli
Department of Computer Science, University of Bologna, Bologna, Italy
e-mail: vittorio.maniezzo@unibo.it,roffilli@csr.unibo.it

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_5,

searched a big search space that could have been much reduced, or having disregarded important information that could have been very effective for constructing good solutions.

Dual information is also tightly connected with the possibility of obtaining good lower bounds (making reference, here and forward, to minimization problems), another element which is not a structural part of current metaheuristics. On the contrary, most mathematical programming literature dedicated to exact methods is strongly based on these elements for achieving the obtained results. There is nothing, though, that limits the effectiveness of dual/bounding procedures to exact methods. There are in fact wide research possibilities both in determining how to convert originally exact methods into efficient heuristics and in designing new, intrinsically heuristic techniques, which include dual information.

In this tutorial we examine a possibility from the second alternative. There are many ways in which bounds can be derived, one of the most effective of these is the use of decomposition techniques [6]. These are techniques primarily meant to exploit the possibility of identifying a subproblem in the problem to solve and to decompose the whole problem in a *master problem* and a *subproblem*, which communicate via dual or dual-related information. The popularity of these techniques derives both from their effectiveness in providing efficient bounds and from the observation that many real-world problems lead themselves to a decomposition.

Unfortunately, despite their prolonged presence in the optimization literature, there is as yet no clear-cut recipe for determining which problems should be solved with decompositions and which are better solved by other means. Clearly, decomposition techniques are foremost candidates for problems which are inherently structured as a master and different subproblems, but it is at times possible to effectively decompose the formulation of a problem which does not show such structure and enjoy advantages. Examples from the literature of effective usage of decomposition techniques (mainly Lagrangean) on single-structure problems include, e.g., set covering [13, 14], set partitioning [3, 32, 12] and crew scheduling [11, 18, 19, 24].

In a previous paper [9] we observed that the general structure of decomposition techniques can be extended from bound computation to include feasible solution construction. According to this, decompositions such as Dantzig-Wolfe, Benders or Lagrangean provide a rich framework for designing metaheuristics. In this work we elaborate this point, showing how the three mentioned approaches can be practically applied to a well-known combinatorial optimization problem, namely the Single Source Capacitated Facility Location Problem.

The structure of the chapter is as follows. In Section 5.2 we introduce the three basic decomposition techniques: Lagrangean relaxation, Dantzig-Wolfe decomposition, and Benders decomposition. Section 5.3 shows, for each of the three methods, how to derive a possible metaheuristic. Section 5.4 introduces the Single Source Capacitated Facility Location Problem, which

will be used for benchmarking the algorithms. Finally, Section 5.5 shows the computational results obtained with our implementation of the proposed metaheuristics.

5.2 Decomposition Methods

This section briefly overviews the three decomposition techniques we will use as a basis for metaheuristics design. These decompositions can be applied to continuous, mixed-integer and pure integer linear programming problems. Since decomposition is a basic operations research topic, which can be found in any mathematical programming textbook, we only present here the basic formulae in the general case of a mixed integer problem. The discussion is for a minimization problem, being trivial to apply it to a maximization one.

The problem to solve, called P, has the following structure:

$$z_P = \min \mathbf{c}_1\mathbf{x} + \mathbf{c}_2\mathbf{y} \tag{5.1}$$
$$s.t.\ \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \geq \mathbf{b} \tag{5.2}$$
$$\mathbf{D}\mathbf{y} \geq \mathbf{d} \tag{5.3}$$
$$\mathbf{x} \geq \mathbf{0} \tag{5.4}$$
$$\mathbf{y} \geq \mathbf{0} \text{ and integer} \tag{5.5}$$

We assume, for ease of presentation, that the feasibility region is non-null and bounded.

5.2.1 Lagrangean Relaxation

Lagrangean relaxation permits to obtain a lower bound to problem P by removing some difficult constraints and by dualizing them into the objective function by means of *Lagrangean penalties*. For example, if in problem P we relax constraints (5.2) using the non-negative Lagrangean penalty vector $\boldsymbol{\lambda}$, we obtain the following formulation LR:

$$z_{LR}(\boldsymbol{\lambda}) = \min \mathbf{c}_1\mathbf{x} + \mathbf{c}_2\mathbf{y} + \boldsymbol{\lambda}(\mathbf{b} - \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{y}) \tag{5.6}$$
$$s.t.\ \mathbf{D}\mathbf{y} \geq \mathbf{d} \tag{5.7}$$
$$\mathbf{x} \geq \mathbf{0} \tag{5.8}$$
$$\mathbf{y} \geq \mathbf{0} \text{ and integer} \tag{5.9}$$

$z_{LR}(\boldsymbol{\lambda})$ is a valid lower bound to the optimal value of P, i.e., $z_{LR}(\boldsymbol{\lambda}) \leq z_P$, for every $\boldsymbol{\lambda} \geq \mathbf{0}$. To identify the penalty vector $\boldsymbol{\lambda}$ that maximizes the lower bound $z_{LR}(\boldsymbol{\lambda})$, we solve the so-called *Lagrangean dual*, which can be

formulated as follows:

$$z_{LR} = \max \{z_{LR}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \geq \mathbf{0}\} \tag{5.10}$$

For solving the Lagrangean dual, an internal *subproblem* LR must be solved for each penalty vector λ. LR is as follows:

$$z_{LR}(\boldsymbol{\lambda}) = \min\ (\mathbf{c}_1 - \boldsymbol{\lambda}\mathbf{A})\mathbf{x} + (\mathbf{c}_2 - \boldsymbol{\lambda}\mathbf{B})\mathbf{y} + \boldsymbol{\lambda}\mathbf{b} \tag{5.11}$$

$$s.t.\ \mathbf{D}\mathbf{y} \geq \mathbf{d} \tag{5.12}$$

$$\mathbf{x} \geq \mathbf{0} \tag{5.13}$$

$$\mathbf{y} \geq \mathbf{0} \text{ and integer} \tag{5.14}$$

If the subproblem is solved to integrality, it is possible that the lower bound provided by z_{LR} is tighter than the linear relaxation of problem P.

Notice that it is possible to add to the LR formulation constraints that are redundant in the original formulation, but that can help the convergence. Moreover, it is sometimes possible to obtain feasible dual solutions directly from the Lagrangean penalties. Approaches based on this property have been used, e.g., to generate reduced problems which consider only the variables of k-least reduced costs (e.g., [11, 12, 24]).

5.2.2 Dantzig-Wolfe Decomposition

Dantzig-Wolfe decomposition [16] is an iterative procedure which successively approximates the linear relaxation of problem P by decomposing it into a sequence of smaller and/or easier *subproblems*. The subproblems dynamically generate the columns of a *master problem* corresponding to the LP relaxation of P.

In order to use the same decomposition as in Section 5.2.1, let F be the feasible region induced by constraints (5.3)–(5.5), i.e. $F = \{(\mathbf{x}, \mathbf{y}) : \mathbf{D}\mathbf{y} \geq \mathbf{d}, \mathbf{x} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0}$ and integer$\}$, which we assume finite and non-null, and let $\{(\mathbf{x}^t, \mathbf{y}^t) : t = 1, \ldots, T\}$ be the set of the extreme points of F. Dantzig-Wolfe proceeds by identifying optimal (with respect to the current cost function) extreme points of F, computed as solutions of a *subproblem*, then passing them to the *master problem* in order to check them against the relaxed constraints, i.e., those not F-defining. The master problem is formulated as a constrained linear combination of the proposed extreme points. After having computed the cost of the best combination of the so far proposed extreme points of F, taking into consideration also the relaxed constraints retained in the master, the subproblem costs are updated, and are computed as reduced costs derived from the dual values of the relaxed constraints. The subproblem is then solved again, to see whether a new, less expensive extreme point can be found.

In the case of problem P, a possible master problem, obtained again by relaxing the "difficult" constraints (5.2) is as follows:

$$z_{MDW} = \min \sum_{t=1}^{T} (\mathbf{c}_1\mathbf{x}^t + \mathbf{c}_2\mathbf{y}^t)\mu_t \tag{5.15}$$

$$s.t. \sum_{t=1}^{T} (\mathbf{A}\mathbf{x}^t + \mathbf{B}\mathbf{y}^t)\mu_t \geq \mathbf{b} \tag{5.16}$$

$$\sum_{t=1}^{T} \mu_t = 1 \tag{5.17}$$

$$\mu_t \geq 0, \qquad t = 1, \ldots, T \tag{5.18}$$

The corresponding subproblem is:

$$z_{SDW}(\mathbf{u}, \alpha) = \min\ (\mathbf{c}_1 - \mathbf{u}\mathbf{A})\mathbf{x} + (\mathbf{c}_2 - \mathbf{u}\mathbf{B})\mathbf{y} - \alpha \tag{5.19}$$

$$s.t.\ \mathbf{D}\mathbf{y} \geq \mathbf{d} \tag{5.20}$$

$$\mathbf{x} \geq \mathbf{0} \tag{5.21}$$

$$\mathbf{y} \geq \mathbf{0} \text{ and integer} \tag{5.22}$$

where $\mathbf{u}$ and α are the dual variables corresponding to constraints (5.16) and (5.17) of the master problem, respectively.

If the subproblem optimal solution $(\mathbf{x}^*, \mathbf{y}^*)$ has a value $z_{SDW}(\mathbf{u}, \alpha) < 0$, we can add the corresponding column $(\mathbf{A}\mathbf{x}^* + \mathbf{B}\mathbf{y}^*)$ of cost $(\mathbf{c}_1\mathbf{x}^* + \mathbf{c}_2\mathbf{y}^*)$ into the master problem, otherwise we have reached the optimal solution of MDW. Notice that subproblem SDW is identical to LR if we replace $\mathbf{u}$ and α with $\boldsymbol{\lambda}$ and $-\boldsymbol{\lambda}\mathbf{b}$, respectively.

At each iteration of the procedure, a valid lower bound to the optimal solution value of the original problem is given by $z_{MDW} + z_{SDW}$ (see [6] for further details). This lower bound is not monotonically nondecreasing. Therefore, we need to maintain the best value obtained through the iterations.

5.2.3 Benders Decomposition

Benders decomposition [8] computes a lower bound to the optimal cost of the original problem by solving a *master problem* which fixes some of its variables. Then, to improve the lower bound, it solves a *subproblem* which adds new constraints to the master.

Let $\mathbf{w}$ and $\mathbf{v}$ be the dual variables of problem P associated to constraints (5.2) and (5.3), respectively. The dual of P is as follows:

$$z_D = \max\ \mathbf{wb} + \mathbf{vd} \tag{5.23}$$
$$s.t.\ \mathbf{wA} \leq \mathbf{c_1} \tag{5.24}$$
$$\mathbf{wB} + \mathbf{vD} \leq \mathbf{c_2} \tag{5.25}$$
$$\mathbf{w} \geq \mathbf{0} \tag{5.26}$$
$$\mathbf{v} \geq \mathbf{0} \tag{5.27}$$

The dual D can be also rewritten as $z_D = \max\{z_{SD}(\mathbf{w}) : \mathbf{wA} \leq \mathbf{c_1}, \mathbf{w} \geq \mathbf{0}\}$, where

$$z_{SD}(\mathbf{w}) = \max\ \mathbf{wb} + \mathbf{vd} \tag{5.28}$$
$$s.t.\ \mathbf{vD} \leq \mathbf{c_2} - \mathbf{wB} \tag{5.29}$$
$$\mathbf{v} \geq \mathbf{0} \tag{5.30}$$

Let $\mathbf{y}$ be the dual variables associated to constraints (5.29). The dual of SD becomes:

$$z_{SP}(\mathbf{w}) = \min\ (\mathbf{c}_2 - \mathbf{wB})\mathbf{y} + \mathbf{wb} \tag{5.31}$$
$$s.t.\ \mathbf{Dy} \geq \mathbf{d} \tag{5.32}$$
$$\mathbf{y} \geq \mathbf{0} \text{ and integer} \tag{5.33}$$

Upon denoting $W = \{\mathbf{w} : \mathbf{wA} \leq \mathbf{c_1}, \mathbf{w} \geq \mathbf{0}\}$ and $Y = \{\mathbf{y} : \mathbf{Dy} \geq \mathbf{d}, \mathbf{y} \geq \mathbf{0} \text{ and integer}\}$, we can rewrite problem D as:

$$z_D = \max_{w \in W} \min_{\mathbf{y} \in Y} (\mathbf{c}_2 - \mathbf{wB})\mathbf{y} + \mathbf{wb}. \tag{5.34}$$

Let $\{\mathbf{w}^t, t = 1, \ldots, T\}$ be the set of the extreme points of W. Since we have assumed the feasible region to be finite and non-null, we have that $z_D = \min_{\mathbf{y} \in \mathbf{Y}} \max_{t=1,\ldots,T} (\mathbf{c}_2 - \mathbf{w}^t\mathbf{B})\mathbf{y} + \mathbf{w}^t\mathbf{b}$, which is equivalent to the following formulation MB:

$$z_{MB} = \min\ z \tag{5.35}$$
$$s.t.\ z \geq (\mathbf{c}_2 - \mathbf{w}^t\mathbf{B})\mathbf{y} + \mathbf{w}^t\mathbf{b}, \quad t = 1, \ldots, T \tag{5.36}$$
$$\mathbf{y} \in \mathbf{Y} \tag{5.37}$$

Problem MB is the *Benders master problem* and constraints (5.36) are the so-called *Benders cuts*. The number of Benders cuts T is usually huge; therefore, the master problem is initially solved considering only a small number T' of Benders cuts, i.e., $T' \ll T$. In order to ascertain whether the solution is already optimal or an additional cut should be added to the master, we need to solve a *subproblem* SB. Since problem D defined in (5.34) is equivalent to:

$$z_D = \min_{\mathbf{y} \in Y} \left(\mathbf{c}_2\mathbf{y} + \max_{w \in W} \mathbf{w}(\mathbf{b} - \mathbf{By}) \right) \tag{5.38}$$

the subproblem SB is:

$$z_{SB}(\mathbf{y}) = \max \mathbf{w}(\mathbf{b} - \mathbf{B}\mathbf{y}) \tag{5.39}$$
$$s.t.\ \mathbf{w}\mathbf{A} \leq \mathbf{c_1} \tag{5.40}$$
$$\mathbf{w} \geq \mathbf{0} \tag{5.41}$$

Notice that a primal solution $(\mathbf{x}, \mathbf{y})$ of problem P, useful for the metaheuristics discussed in Section 5.3, can be obtained by the dual of SB defined as:

$$z_{SP}(\mathbf{y}) = \min \mathbf{c}_1\mathbf{x} \tag{5.42}$$
$$s.t.\ \mathbf{A}\mathbf{x} \leq \mathbf{b} - \mathbf{B}\mathbf{y} \tag{5.43}$$
$$\mathbf{x} \geq \mathbf{0} \tag{5.44}$$

Also for problems MB and SB, it is possible to add constraints that are redundant in the original formulation, but can help convergence.

It is interesting to show that also for Benders decomposition we can have a subproblem equivalent to the ones of Lagrangean relaxation and Dantzig-Wolfe decomposition. In fact, if the dual D, (5.23)–(5.27), is rewritten as:

$$z_D = \max \{z_{SD'}(\mathbf{w}) : \mathbf{w} \geq \mathbf{0}\} \tag{5.45}$$

where

$$z_{SD'}(\mathbf{w}) = \max \mathbf{w}\mathbf{b} + \mathbf{v}\mathbf{d} \tag{5.46}$$
$$s.t.\ \mathbf{0} \leq \mathbf{c_1} - \mathbf{w}\mathbf{A} \tag{5.47}$$
$$\mathbf{v}\mathbf{D} \leq \mathbf{c_2} - \mathbf{w}\mathbf{B} \tag{5.48}$$
$$\mathbf{v} \geq \mathbf{0} \tag{5.49}$$

the dual of SD′ is identical to subproblem LR, defined by (5.11)–(5.14), after replacing the penalty vector $\boldsymbol{\lambda}$ with $\mathbf{w}$.

5.3 Metaheuristics Derived from Decompositions

In this section we show how metaheuristic frameworks can be directly derived from the three decomposition methods previously described. Notice that the proposed algorithms are not the only ones that could be derived from the used decompositions, but they represent reasonable frameworks, which we have already used with success on different problems. We hope that this chapter may serve as a means to foster research on different or more general metaheuristic frameworks, including other approaches deriving from decomposition techniques.

5.3.1 A Lagrangean Metaheuristic

The literature is rich with heuristics based on the Lagrangean decomposition structure outlined above. An excellent introduction to the whole topic of Lagrangean relaxation, and of related heuristics, can be found in [7]. A general structure of a Lagrangean heuristic, common to most applications, is given in Algorithm 1.

Algorithm 1: LagrHeuristic

```
1 identify an "easy" subproblem LR(λ)
2 repeat
3 |   solve subproblem LR(λ) obtaining solution x
4 |   check for unsatisfied constraints
5 |   update penalties λ
6 |   construct problem solution using x and λ
7 until (end_condition) ;
```

This pseudocode is obviously underspecified for a direct application, being at an abstraction level where metaheuristics are usually presented. However, notice that this structure already shows the essential ingredients of a metaheuristic, i.e., it is "an iterative master process that guides and modifies the operations of a subordinate heuristic" at Step 6.

Steps 1 and 3 are problem-dependent, such as neighborhood definition or crossover implementation in other contexts. Step 4 is trivial, while Step 5 can be implemented by means of any state-of-the-art technique, usually subgradient optimization or bundle methods. Moreover, some of these techniques have been proved to converge not only to the optimal $\boldsymbol{\lambda}$, but also to the optimal $\mathbf{x}$ of the linear relaxation (see Sherali and Choi [29] and Barahona and Anbil [4]), thereby possibly providing a particularly "intelligent" starting point for Step 6.

5.3.2 A Dantzig-Wolfe Metaheuristic

As for any metaheuristic, also for Dantzig-Wolfe we can propose a general structure that will have to be detailed in some of its steps in order to apply it to specific problems. Here, we propose one possible effective structure, but again, alternative ones are possible.

The master problem MDW should be defined to be easy to solve to optimality, while the subproblem SDW can be difficult and it could be needed to solve it heuristically. The proposed pseudocode for algorithm DWHeuristic tries to generate feasible solutions making use of the dual solutions $(\mathbf{u}, \alpha)$ provided by MDW and of the primal solution $(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{T} (\mathbf{x}^t, \mathbf{y}^t)\mu_t$

Algorithm 2: DWHEURISTIC

```
1  identify a master MDW and an "easy" subproblem SDW(u,α), set T=0
2  repeat
3      solve master problem MDW
4      given the solution μ of MDW define (x, y) = Σ_{t=1}^{T} (x^t, y^t) μ_t
5      solve problem SDW(u,α), where (u,α) is the dual solution of MDW
6      construct feasible solutions using (x, y) and/or (u,α), generated by MDW,
7          and/or (x', y'), generated by SDW(u,α)
8      if (no more columns can be added) then
9          STOP
10     else
11         set T = T + 1
12         add the column (x', y') generated at step 5
13 until (end_condition) ;
```

and $(\mathbf{x}', \mathbf{y}')$ generated by solving MDW and SDW($\mathbf{u}$,α), respectively. However, it is possible to include other local search algorithms, based on different neighborhoods. For example, we can generate a feasible solution using the solutions $(\mathbf{x}^t, \mathbf{y}^t)$ associated to the columns of MDW with $\mu_t > 0$ in its current solution.

5.3.3 A Benders Metaheuristic

The identification of a common structure for Benders based heuristics is more difficult than for Lagrangean or Dantzig-Wolfe ones, since the proposals in the literature vary much, and usually Benders decomposition is used in a very problem-dependent fashion. We propose here one possible structure, which already proved effective, but again, alternative ones are possible.

The structure can be applied both to MIP problems, as sketched in Section 5.2, and to pure IP problems. The subproblem SP (see Equation (5.42)) could be defined over integer or binary variables, in both cases it is necessary to use its linear relaxation in order to obtain its dual SB (Equation (5.39)).

Taking into account the intrinsic difficulty of both MB and SB, we propose to consider solving them both heuristically. The effect of solving heuristically MB at step 3 is that it is not guaranteed to produce a lower bound to problem P. When a lower bound is needed, MB must be solved to optimality, or approximated from below. Notice, however, that the main purpose of MB is to produce alternative $\mathbf{y}$ sets, of possibly increasing qualities, and this can be effectively accomplished by heuristic solutions. Step 5 provides an upper bound, i.e., a feasible solution, to the whole problem. Step 6 finds a lower bound to the problem obtained by fixing the $\mathbf{y}$ variables.

Algorithm 3: BendHeuristic

```
1  identify a master MB and an "easy" subproblem SB(y), set T = 0
2  repeat
3      solve (heuristically) master problem MB obtaining the solution (z, y)
4      if (x are requested to be integer) then
5          solve (heuristically) master problem MB obtaining the solution (z, y)
6      solve problem SB(y) obtaining the dual solution w
7      if (no more columns can be added) then
8          STOP
9      else
10         set T = T + 1
11         add to MB the Benders cut generated by problem SB(y)
12 until (end_condition) ;
```

The terminating condition at Step 7 depends on whether the master is solved heuristically or to optimality. In this last case, the condition would be "**if** $z^t \geq z_d$", which in fact implies the impossibility of generating new cuts. However, in a heuristic context such as admitted by Steps 3 and 5, new cuts could be further generated, which could prove useful for continuing search.

5.4 Single Source Capacitated Facility Location

The algorithms presented in Section 5.3 are meant as metaheuristics. They are relatively simple, yet effective and robust approaches. To get state-of-the-art results some sophisticated elements are needed, for these as for any other metaheuristic. However, a straightforward application of these pseudocodes already produces results, which are close to the state-of-the-art. In order to show the robustness and the ease to arrive to fully-defined, problem-specific codes, we report in this section on the application of each proposed approach to the Single Source Capacitated Facility Location Problem (SCFLP).

The SCFLP is a well-known problem that arises in many applications, from clustering problems in data mining to networks design. The problem asks to locate a number of facilities (e.g., plants, warehouses or hubs), that must provide a service to a set of customers, minimizing a global cost. The cost includes fixed charges for opening the facilities and service costs for satisfying customer demands.

Let $J = \{1, \ldots, n\}$ be the index sets of customers and $I = \{1, \ldots, m\}$ the index set of possible facility locations. Each customer j has an associated demand, q_j, that must be served by a single facility; a facility located at site i has an overall capacity of Q_i. The costs are composed of a cost c_{ij} for supplying the demand of a customer j from a facility established at location i and of a fixed cost, f_i, for opening a facility at location i. Let x_{ij}, $i = 1, \ldots, m$,

$j = 1, \ldots, n$, be binary variables such that $x_{ij} = 1$ if customer j is assigned to a facility located at i, 0 otherwise, and let y_i, $i = 1, \ldots, m$, be binary variables such that $y_i = 1$ if a facility is located at site i, 0 otherwise.

A mathematical formulation of the SCFLP is as follows:

$$z_{SCFLP} = \min \sum_{i \in I, j \in J} c_{ij} x_{ij} + \sum_{i \in I} f_i y_i \tag{5.50}$$

$$s.t. \sum_{i \in I} x_{ij} = 1, \qquad j \in J \tag{5.51}$$

$$\sum_{j \in J} q_j x_{ij} \leq Q_i y_i, \qquad i \in I \tag{5.52}$$

$$x_{ij} \in \{0, 1\}, \qquad i \in I, j \in J \tag{5.53}$$

$$y_i \in \{0, 1\}, \qquad i \in I \tag{5.54}$$

The objective function (5.50) asks to minimize the sum of fixed and service costs. Assignment constraints (5.51) ensure that all customers are serviced by exactly one facility; knapsack constraints (5.52) are the facility capacity constraints and, finally, (5.53) and (5.54) are the integrality constraints.

SCFLP is an NP-hard problem and often the optimal value of its LP relaxation, obtained by removing the integrality constraints, is much worse than the optimum integer solution value. In order to improve the optimal value of the LP-relaxation, as suggested in [20], we can add the following additional constraints, redundant in SCFLP:

$$x_{ij} - y_i \leq 0, \qquad \text{for each } i \in I \text{ and } j \in J \tag{5.55}$$

Given its simple structure, the SCFLP has often been used for benchmarking new approaches. Some variants of it exist, the most studied one permits a split assignment of customers to location, thus relaxing constraints (5.53) to $x_{ij} \geq 0, i \in I, j \in J$. Most approximation results, such as Chudak and Shmoys's 3-approximation algorithm [15], refer to this problem version. Closely related problems are also the Capacitated p-median and the Generalized Assignment Problems. Several exact approaches have been proposed for the SCFLP, one of the best known being [25], where a branch and bound scheme based on a partitioning formulation is proposed. However, exact methods do not scale up to large instance sizes.

Large instances have been tackled by means of different kinds of heuristics, from very large scale neighborhood (VLSN) search [2] to reactive GRASP and tabu search [17]. Extensive research has been devoted to Lagrangean heuristics for the SCFLP. Most authors start by relaxing assignment constraints, obtaining a Lagrangean subproblem which separates into n knapsack problems, one for each facility, whose combined solutions provide a lower bound to the problem [5, 26, 31, 20, 28]. However, different relaxations have also been used. Klincewicz and Luss [21] relax the capacity constraints (5.52),

thereby obtaining as Lagrangean subproblem an uncapacitated facility location problem, which is solved heuristically. Beasley [7] and Agar and Salhi [1] relax both the assignment and the capacity constraints, and obtain a very robust solution approach, which provides good quality solutions to a number of different location problems, including p-median, uncapacitated, capacitated and single source facility location problems.

Having introduced the basic techniques and the problem they have been applied to, we move on describing how the basic pseudocodes for the Lagrangean, Dantzig-Wolfe and Benders metaheuristics can be specialized for the SCFLP.

5.4.1 Solving the SCFLP with a Lagrangean Metaheuristic

We present here a very straightforward application of LagrHeuristic to the SCFLP. The resulting algorithm is not enough to produce edge-level results, but it shows that already by means of such a simple code it is possible to get quite good performance. The steps of LagrHeuristic for the SCFLP can be specified as follows. (Note that the step numbers refer to the lines in the pseudocode of the metaheuristic.)

Step 1: Identify an "easy" subproblem LR. The relaxation of the assignment constraints (5.51) in problem SCFLP yields the following problem.

$$z_{LR}(\boldsymbol{\lambda}) = \min \sum_{i\in I, j\in J} (c_{ij} - \lambda_j)x_{ij} + \sum_{i\in I} f_i y_i + \sum_{j\in J} \lambda_j \tag{5.56}$$

$$s.t. \sum_{j\in J} q_j x_{ij} \le Q_i y_i, \qquad i \in I \tag{5.57}$$

$$x_{ij} \in \{0,1\}, \qquad i \in I, j \in J \tag{5.58}$$

$$y_i \in \{0,1\}, \qquad i \in I \tag{5.59}$$

where λ_j, $j \in J$, are unrestricted penalties.

Step 3: Solve subproblem LR. Problem LR decomposes naturally into $|I|$ knapsack problems, with objective function $\sum_{i\in I}\left(\sum_{j\in J}(c_{ij} - \lambda_j)x_{ij} + f_i y_i\right)$. Thus, for each $i \in I$ for which $\sum_{j\in J}(c_{ij} - \lambda_j)x_{ij} < -f_i$, the corresponding y_i is set to 1, otherwise to 0.

Step 4: Check for unsatisfied constraints. The solution of LR can have customers assigned to multiple or to no location. This can be determined by direct inspection.

Step 5: Update penalties $\boldsymbol{\lambda}$. We used a standard subgradient algorithm [26] for updating penalties.

Step 6: construct problem solution using $\mathbf{x}$ *and* $\boldsymbol{\lambda}$. Let $\bar{I}$ be the set of locations chosen in the solution obtained at Step 3. The SCFLP becomes a Generalized Assignment Problem (GAP) as follows:

$$z_{GAP} = \min \sum_{i \in \bar{I}, j \in J} c_{ij} x_{ij} \tag{5.60}$$

$$s.t. \sum_{i \in \bar{I}} x_{ij} = 1, \qquad j \in J \tag{5.61}$$

$$\sum_{j \in J} q_j x_{ij} \leq Q_i, \qquad i \in \bar{I} \tag{5.62}$$

$$x_{ij} \in \{0, 1\}, \qquad i \in \bar{I}, j \in J \tag{5.63}$$

This is still an NP-hard problem, but efficient codes exist to solve it, which we did once per Lagrangean iteration (see the subsequent computational results section for further details).

We formulate the GAP using the original costs $\{c_{ij}\}$ instead of the penalized costs $\{c_{ij} - \lambda_j\}$, which could seem to be an obvious bonus granted by using the Lagrangean relaxation in a heuristic context. This is because in this case, having fixed the set of chosen locations $\bar{I}$, solving the GAP to optimality generates the best possible solution. However, in other circumstances, we can take advantage of using penalized (thus dual-related) costs instead of the original ones (e.g., the fully distributed Lagrangean metaheuristic for a P2P Overlay Network Design Problem described in [10, 22]) obtaining a considerable computational advantage.

Notice that for some iterations, Step 3 may provide a set of locations $\bar{I}$ for which the GAP is unfeasible. In this case no feasible SCFLP solution is generated and LagrHeuristic simply goes on.

5.4.2 Solving the SCFLP with a Dantzig-Wolfe Metaheuristic

We have a number of possibilities to decompose our model for the SCFLP. Among them we chose to decompose the problem in such a way as to have a subproblem equivalent to LR, defined for the Lagrangean relaxation described in the previous subsection. The specific steps of DWHeuristic for the SCFLP result as follows.

Step 1: Identify a master MDW and an "easy" subproblem SDW. A possible Dantzig-Wolfe decomposition of the SCFLP maintains the assignment

constraints in the master problem:

$$z_{MDW} = \min \sum_{k=1}^{t} \left(\sum_{i \in I, j \in J} c_{ij} x_{ij}^k + \sum_{i \in I} f_i y_i^k \right) \lambda_k \tag{5.64}$$

$$s.t. \sum_{k=1}^{t} \left(\sum_{i \in I} x_{ij}^k \right) \lambda_k = 1, \qquad j \in J \tag{5.65}$$

$$\sum_{k=1}^{t} \lambda_k = 1 \tag{5.66}$$

$$\lambda_k \geq 0, \qquad k = 1, \ldots, t \tag{5.67}$$

and the subproblem is:

$$z_{SDW}(\mathbf{u}, \alpha) = \min \sum_{i \in I, j \in J} (c_{ij} - u_j) x_{ij} + \sum_{i \in I} f_i y_i - \alpha \tag{5.68}$$

$$s.t. \sum_{j \in J} q_j x_{ij} \leq Q_i y_i, \qquad i \in I \tag{5.69}$$

$$x_{ij} \in \{0, 1\}, \qquad i \in I, j \in J \tag{5.70}$$

$$y_i \in \{0, 1\}, \qquad i \in I \tag{5.71}$$

where u_j and α are the dual variables of the master problem corresponding to constraints (5.65) and (5.66), respectively.

Step 3: Solve master problem MDW. As the master problem is relatively easy to solve, we solve it to optimality at each iteration.

Step 5: Solve subproblem SDW. The x_{ij} and y_i are required to be integer, but subproblem SDW is equivalent to LR, (5.56)–(5.59), and it can be decomposed into $|I|$ knapsack problems, with objective function $\sum_{i \in I} \left(\sum_{j \in J} (c_{ij} - u_j) x_{ij} + f_i y_i \right)$. Thus, for each $i \in I$ for which $\sum_{j \in J} (c_{ij} - u_j) x_{ij} < -f_i$, the corresponding y_i is set to 1, otherwise to 0.

Step 6: Construct a feasible solution. We generate a feasible SCFLP solution using the same procedure used for the Lagrangean metaheuristic by solving problem GAP, (5.60)–(5.63), defined according to the SDW solution.

Step 8: Stop condition. If $z_{SDW}(\mathbf{u}, \alpha) \geq 0$ we stop because we have reached the optimal solution of the master problem MDW. Otherwise, we add the new column generated by SDW to the master problem MDW and we go on.

5.4.3 Solving the SCFLP with a Benders Metaheuristic

Also a Benders metaheuristic approach offers a number of possibilities to decompose our model and to generate feasible solutions for the SCFLP. The implementation of BendHeuristic that we have chosen is as follows.

Step 1: Identify a master MB and an "easy" subproblem SP. A possible Benders decomposition of SCFLP involves keeping in the master the decision of which facilities to open, and assigning clients to open facilities as a subproblem. The subproblem is therefore a GAP again.

More in detail, the master problem is:

$$z_{MB} = min \sum_{i \in I} f_i y_i + z_{SP}(\mathbf{y}) \tag{5.72}$$

$$s.t.\ y_i \in \{0,1\}, \qquad i \in I \tag{5.73}$$

and the subproblem becomes:

$$z_{SP}(\mathbf{y}) = min \sum_{i \in I, j \in J} c_{ij} x_{ij} \tag{5.74}$$

$$s.t. \sum_{i \in I} x_{ij} = 1, \qquad j \in J \tag{5.75}$$

$$\sum_{j \in J} q_j x_{ij} \leq Q_i y_i, \qquad i \in I \tag{5.76}$$

$$x_{ij} \leq y_i, \qquad i \in I, j \in J \tag{5.77}$$

$$x_{ij} \in \{0,1\}, \qquad i \in I, j \in J \tag{5.78}$$

where constraints (5.77) are considered only if the integrality constraints (5.73) are relaxed.

Step 3: Solve master problem MB. As the master problem, even though NP-hard after the addition of Bender's cuts, was relatively easy to solve for the benchmark instances from the literature, we solved it to optimality at each iteration.

Step 6: Solve subproblem SP. The x_{ij} are required to be integer, but the subproblem is the same GAP we met in Subsection 5.4.1. The same considerations apply.

Step 11: Add to MB the Benders cut generated by problem SB. To get the subproblem's dual we relaxed constraints (5.78) into $x_{ij} \geq 0$, $i \in I, j \in J$. After associating dual variables w'_j, $j \in J$, to constraints (5.75), w''_i, $i \in I$, to constraints (5.76) and w'''_{ij}, $i \in I$, $j \in J$, to constraints (5.77), problem SB becomes:

$$z_{SB}(\mathbf{y}) = max \sum_{j\in J} w'_j + \sum_{i\in I} Q_i y_i w''_i + \sum_{i\in I}\sum_{j\in J} y_i w'''_{ij} \tag{5.79}$$

$$s.t.\ w'_j + q_j w''_i + w'''_{ij} \le c_{ij}, \qquad i \in I, j \in J \tag{5.80}$$

$$w''_i \le 0, \qquad i \in I \tag{5.81}$$

$$w'''_{ij} \le 0, \qquad i \in I, j \in J \tag{5.82}$$

The master formulation, which includes the added cut, is as follows:

$$z_{MB} = min\ z$$

$$s.t.\ z \ge \sum_{i\in I}\left(f_i + Q_i w''_i + \sum_{j\in J} w'''_{ij}\right) y_i + \sum_{j\in J} w'_j \tag{5.83}$$

$$y_i \in \{0,1\}, \qquad i \in I \tag{5.84}$$

5.5 Computational Results

We implemented the above described algorithms in C# and Fortran, this last was used by linking algorithms MT1R for solving knapsack problems and MTHG for getting a heuristic solution of GAP problems [23] (codes can be freely downloaded from the page `http://www.or.deis.unibo.it/knapsack.html`). The code was run on a 1.7 GHz laptop with 1Gb of RAM and .NET framework 2.0. Ilog CPLEX 11.1 was used as LP and MIP solver where required.

The benchmark instances are those used by Holmberg et al. [20]; they consist of 71 instances whose size ranges from 50 to 200 customers and from 10 to 30 candidate facility locations. The instances are divided into four subsets. Set 1 has customers and locations with coordinates randomly generated in the interval $[10, 200]$, problems p1 to p12 have 50 customers and 10 possible locations, problems p13 to p24 have 50 customers and 20 possible locations. Set 2 has locations generated in the interval $[10, 300]$. The assignment costs are based on a vehicle routing problem cost distribution (see [20] for details). Set 3 is based on vehicle routing test problems used by Solomon [30], while set 4 is generated as set 1 but the number of potential locations is 30 and the number of customers is 200.

In this section we present computational results for the three proposed metaheuristic procedures, namely, LagrHeuristic, DWHeuristic and BendHeuristic, and compare them with those obtained by the "*dfs*" variant of the VLSN heuristic proposed by [2], which is the best performing metaheuristic algorithm known for the SCFLP. The CPU times reported for *dfs* have been obtained on a PC with an Athlon/1200Mhz processor and 512 Mb RAM, under RedHat Linux 7.1.

Table 5.1 Computational results obtained with procedure LagrHeuristic.

Problem	Lagrangean Metaheuristic					dfs	
Sets	G_{LP}	G_H	G_L	T_{Best}	T_{Tot}	G_{dfs}	T_{dfs}
p1-p24 avg	0.79	0.01	0.12	0.16	4.38	0.00	0.54
p1-p24 max	2.19	0.06	0.66	0.85	14.80	0.00	1.85
p25-p40 avg	0.77	0.55	0.69	1.51	51.04	0.13	12.67
p25-p40 max	2.02	2.95	1.86	10.77	107.21	0.79	34.08
p41-p55 avg	0.84	0.30	0.31	0.91	10.46	0.03	1.62
p41-p55 max	2.00	2.02	1.86	3.68	31.55	0.18	5.47
p56-p71 avg	0.57	0.21	0.57	27.27	475.15	0.02	15.97
p56-p71 max	2.28	1.06	1.95	201.74	1731.80	0.14	46.60

Let z_{MIP} and z_{LP} be the optimal solutions of problem SCFLP, (5.50)–(5.54), and of its LP relaxation, respectively. Let z_{UB} and z_{LB} be the upper and the lower bounds provided by the proposed procedures, respectively. In Tables 5.1, 5.2 and 5.3, for each set of test instances, we report the following average and maximum values:

G_{LP}: the percentage gap between the optimal MIP solution and the optimal LP solution, i.e., $G_{LP} = \frac{z_{MIP} - z_{LP}}{z_{MIP}} \times 100$;
G_H: the percentage gap between the heuristic solution provided by the proposed procedure and the optimal MIP solution, i.e., $G_H = \frac{z_{UB} - z_{MIP}}{z_{MIP}} \times 100$;
G_L: the percentage gap between the lower bound provided by the proposed procedure and the optimal MIP solution, i.e., $G_L = \frac{z_{MIP} - z_{LB}}{z_{MIP}} \times 100$;
T_{Best}: the computing time in seconds required by the proposed procedure to reach the best heuristic solution found;
T_{Tot}: the total computing time in seconds required by the proposed procedure;
G_{dfs}: the percentage distance from optimality of *dfs*;
T_{dfs}: the CPU time in seconds taken by *dfs*.

5.5.1 Lagrangean Metaheuristic

Procedure LagrHeuristic was used with the α subgradient step control parameter (see [27]) initially set to 0.5, and multiplied by 0.9 when five consecutive non-improving iterations were detected. LagrHeuristic terminated either when an optimal solution was found, i.e., when $z_{LB} = z_{UB}$, when 5000 subgradient iterations were made, or when a time limit of 3600 seconds was reached.

The computational results for procedure LagrHeuristic are reported in Table 5.1. Figure 5.1 shows the evolution of the upper bound z_{UB} and of the lower bounds z_{LB} when LagrHeuristic is applied to instance *p11*. Proce-

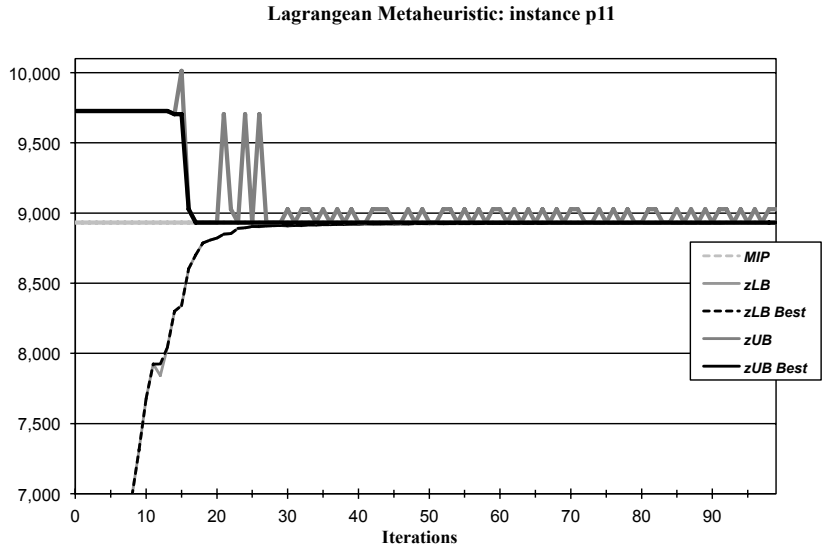


Fig. 5.1 Upper and lower bounds evolution of LagrHeuristic for the instance *p11*.

dure LagrHeuristic shows on all test problems a performance qualitatively similar to the one reported in Figure 5.1.

As repeatedly pointed out, the results we report here are not for showing that we have the best heuristic in the literature, but for showing that even a straightforward implementation of algorithm LagrHeuristic can get close to the state-of-the-art. This is apparent on Table 5.1 where there are not big differences with respect to *dfs* and where the existing gap is mainly due to few instances. It would be rather easy to close that gap by means of some trick on the subgradient algorithm, such as an α-restart or an adaptive anneal (not to mention a local search on the upper bound), but again, this would obfuscate our point.

We mention here again how the inclusion of dual information into the metaheuristic permits to determine the quality of the best solution found, and possibly its optimality. In our case, out of the 71 instances, 3 could be solved to optimality by the subgradient alone, which evolved weights that lead to the satisfaction also of the relaxed constraints, while 21 other ones were solved to proven optimality since the lower and the upper bound converged to the same cost. In all these cases the computation terminated before the maximum available CPU time, an option which is not available for primal-only heuristics.

Table 5.2 Computational results obtained with procedure DWHeuristic.

Problem	Dantzig-Wolfe Metaheuristic					dfs	
Name	G_{LP}	G_H	G_L	T_{Best}	T_{Tot}	G_{dfs}	T_{dfs}
p1-p24 avg	0.79	0.04	0.14	11.39	825.22	0.00	0.54
p1-p24 max	2.19	0.76	0.67	32.20	2558.42	0.00	1.85
p25-p40 avg	0.77	0.55	4.83	1096.97	3581.77	0.13	12.67
p25-p40 max	2.02	2.95	12.48	2028.75	3600.72	0.79	34.08
p41-p55 avg	0.84	0.42	0.60	231.95	2793.80	0.03	1.62
p41-p55 max	2.00	2.02	2.48	1246.37	3600.96	0.18	5.47
p56-p71 avg	0.57	9.19	50.19	2875.35	3600.47	0.02	15.97
p56-p71 max	2.28	34.81	100.00	3555.17	3603.10	0.14	46.60

5.5.2 Dantzig-Wolfe Metaheuristic

We initialized the master problem by adding a column corresponding to a dummy facility with a sufficient capacity to serve all customers, but with a fixed cost equal to a known upper bound to the optimal solution cost.

Procedure DWHeuristic terminates when no further columns can be added to the master problem. However, since the convergence can be slow, procedure DWHeuristic was also stopped when 20000 columns were generated or when a time limit of 3600 seconds was reached.

The computational results reported in Table 5.2 show that the convergence of our basic DWHeuristic is slow and is not competitive with the Lagrangean metaheuristic. This behavior is mainly due to the large number of iterations required to obtain a good lower bound and, building on it, good solutions. Figure 5.2 shows a trace in the case of instance *p11*, where about 700 iterations are required to reach a good primal solution and about 1100 iterations to reach a good lower bound. Figure 5.2 confirms the convergence of upper and lower bounds.

DWHeuristic finds difficulties in solving set 2 and set 4, where the given time limit is not enough to provide a sufficiently good lower bound. For set 4 the average gap between the primal solution and the lower bound is unsatisfactory.

Clearly this basic schema is not competitive and some modifications are required. For example, our basic implementation of DWHeuristic can be improved by adding more columns at each iteration and/or solving heuristically the subproblem SDW, given by (5.68)–(5.71).

5.5.3 Benders Metaheuristic

We initialize the BendHeuristic in the same way as the master problem in the DWHeuristic. Procedure BendHeuristic terminates when either no

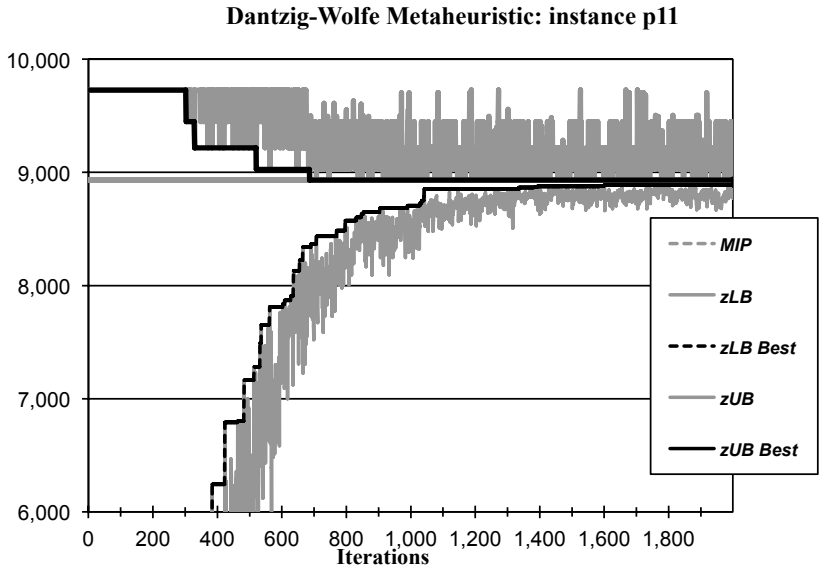


Fig. 5.2 Upper and lower bounds evolution of DWHeuristic for the instance *p11*.

further cuts can be added to the master problem, or when 5000 cuts were added or when a time limit of 3600 seconds was reached.

The computational results reported in Table 5.3 show a lower competitiveness of the basic BendHeuristic, when compared to its Lagrangean counterpart. The basic BendHeuristic outperforms procedure DWHeuristic, requiring less computing time to reach upper and lower bounds of similar quality. Figure 5.3 shows a trace in the case of instance p11, where it is evident that a good primal solution is generated quite quickly but the convergence of the lower bound is slow.

BendHeuristic finds difficulties in solving set 3 and particularly set 4, which has a cost structure that makes the master hard to solve when cuts start to be added. Clearly this basic schema is not competitive on these instances, more sophisticated considerations are required.

However, we believe that, since research on Benders based heuristics counts much less contributions than for instance Lagrange based ones, there is a wide room available for gaining insight on how to improve this basic functioning. For example, in our basic implementation of BendHeuristic we had the master solved to optimality, while it would be worthwhile to solve the master only heuristically.

Table 5.3 Computational results obtained with procedure BendHeuristic.

Problem	Benders Metaheuristic					dfs	
Name	G_{LP}	G_H	G_L	T_{Best}	T_{Tot}	G_{dfs}	T_{dfs}
p1-p24 avg	0.79	0.04	4.26	0.96	1612.28	0.00	0.54
p1-p24 max	2.19	0.76	13.30	6.90	3616.98	0.00	1.85
p25-p40 avg	0.77	0.55	0.34	17.26	2634.50	0.13	12.67
p25-p40 max	2.02	2.95	0.71	48.13	3602.35	0.79	34.08
p41-p55 avg	0.84	0.58	19.80	214.82	1781.79	0.03	1.62
p41-p55 max	2.00	2.02	68.66	1584.98	3611.57	0.18	5.47
p56-p71 avg	0.57	2.89	53.94	277.85	3636.65	0.02	15.97
p56-p71 max	2.28	14.20	80.52	1576.34	3836.41	0.14	46.60

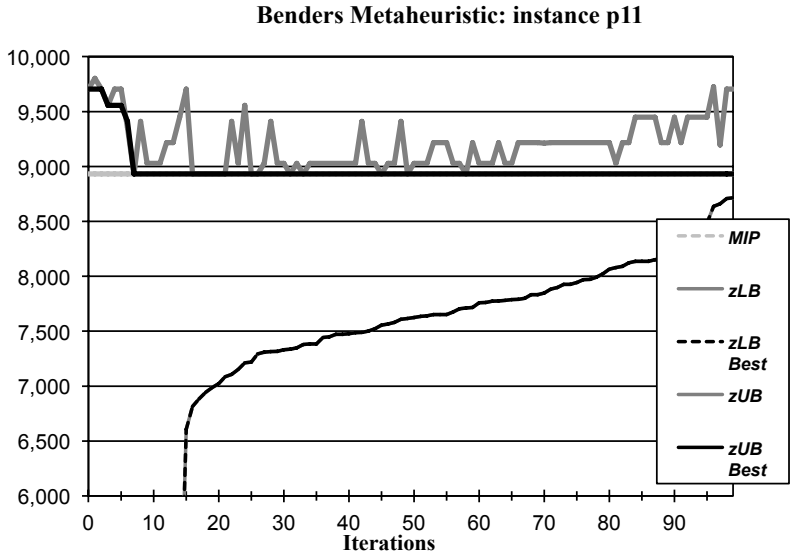


Fig. 5.3 Upper and lower bounds evolution of BendHeuristic for the instance *p11*.

5.6 Conclusions

This tutorial has shown a possibility to derive metaheuristic frameworks from the three main decomposition techniques from the literature, namely Lagrangean, Benders and Dantzig-Wolfe. This is an example, expanding a proposal first published in [9], of how techniques, originally designed for exact methods, could be included in a purely metaheuristic structure which shows the usual properties of simplicity, robustness and effectiveness.

The main point behind our argument is that research on metaheuristic methods should include elements from the mathematical programming literature in order to get a possibility to overcome the current computational limits, whenever these limits are felt to diminish the practical effectiveness of the available procedures.

We believe that the principal contribution of mathematically elaborate techniques comes from the use of bounds to the cost of optimal solutions and from dual information, two elements that can greatly help in directing search for better-than-current solutions and for determining the quality of the results achieved at any moment during search.

The computational results reported in Section 5.5 show that the heuristic solutions provided by the proposed metaheuristics can be of good quality even when the used dual information corresponds to a lower bound far from the optimal solution. Therefore, we can have good results also when the convergence is slow, with the only disadvantage of failing in reliably evaluating the quality. In the proposed frameworks it is mandatory to solve to optimality the subproblem. This can be a serious limitation when the relaxed problem is still difficult and a valid lower bound is not produced. In this case we can further relax the problem until the resulting problem is computationally tractable. However, this is an interesting issue that deserves further investigations to identify other approaches able to overcome all difficulties.

Research on how metaheuristics should make a strong point of mathematical modules is still at an embryonal level. We hope that this tutorial may help in fostering research along this line, that we believe to be promising.

References

1. M. Agar and S. Salhi. Lagrangean heuristics applied to a variety of large capacitated plant location problems. *Journal of the Operational Research Society*, 49:1072–1084, 1998.
2. R.K. Ahuja, J.B. Orlin, S. Pallottino, M.P. Scaparra, and M.G. Scutellà. A multi-exchange heuristic for the single source capacitated facility location problem. *Management Science*, 50(6):749–760, 2003.
3. A. Atamtürk, G. Nemhauser, and M.W.P. Savelsbergh. A combined Lagrangian, linear programming, and implication heuristic for large-scale set partitioning problems. *Journal of Heuristics*, 1:247–259, 1996.
4. F. Barahona and R. Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87:385–399, 2000.
5. J. Barcelo and J. Casanovas. A heuristic Lagrangean algorithm for the capacitated plant location problem. *European Journal of Operational Research*, 15:212–226, 1984.
6. M.S. Bazaraa, J. Jarvis, and H.D. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, 1990.
7. J.E. Beasley. Lagrangean relaxation. In C.R. Reeves, editor, *Modern heuristic techniques for combinatorial problems*, pages 243–303. Blackwell Scientific Publ., 1993.
8. J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:280–322, 1962.

9. M.A. Boschetti and V. Maniezzo. Benders decomposition, Lagrangean relaxation and metaheuristic design. *Journal of Heuristics*, 15(3):283–312, 2009.
10. M.A. Boschetti, V. Maniezzo, and M. Roffilli. A fully distributed Lagrangean solution for a P2P overlay network design problem. *Submitted for publication*, 2009.
11. M.A. Boschetti, A. Mingozzi, and S. Ricciardelli. An exact algorithm for the simplified multi depot crew scheduling problem. *Annals of Operations Research*, 127:177–201, 2004.
12. M.A. Boschetti, A. Mingozzi, and S. Ricciardelli. A dual ascent procedure for the set partitioning problem. *Discrete Optimization*, 5(4):735–747, 2008.
13. A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations Research*, 47:730–743, 1999.
14. S. Ceria, P. Nobili, and A. Sassano. A Lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81:215–228, 1995.
15. F.A. Chudak and D.B. Shmoys. Improved approximation algorithms for a capacitated facility location problem. In *Proc. 10th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages S875–S876, 1999.
16. G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.
17. H. Delmaire, J.A. Diaz, E. Fernandez, and M. Ortega. Reactive GRASP and tabu search based heuristics for the single source capacitated plant location problem. *INFOR*, 37:194–225, 1999.
18. R. Freling, D. Huisman, and A.P.M. Wagelmans. Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, 6:63–85, 2003.
19. K.L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39:657–682, 1993.
20. K. Holmberg, M. Ronnqvist, and D. Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113:544–559, 1999.
21. J. Klincewicz and H. Luss. A Lagrangean relaxation heuristic for capacitated facility location with single-source constraints. *Journal of the Operational Research Society*, 37:495–500, 1986.
22. V. Maniezzo, M.A. Boschetti, and M. Jelasity. A fully distributed Lagrangean metaheuristic for a P2P overlay network design problem. In *Proceedings of the 6th Metaheuristics International Conference (MIC 2005)*, Vienna, Austria, 2005.
23. S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley, 1990.
24. A. Mingozzi, M.A. Boschetti, S. Ricciardelli, and L. Bianco. A set partitioning approach to the crew scheduling problem. *Operations Research*, 47:873–888, 1999.
25. A. Neebe and M. Rao. An algorithm for the fixed-charge assigning users to sources problem. *Journal of the Operational Research Society*, 34:1107–1113, 1983.
26. H. Pirkul. Efficient algorithm for the capacitated concentrator location problem. *Computers & Operations Research*, 14:197–208, 1987.
27. B.T. Polyak. Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*, 9:14–29, 1969.
28. M. Ronnqvist, S. Tragantalerngsak, and J. Holt. A repeated matching heuristic for the single source capacitated facility location problem. *European Journal of Operational Research*, 116:51–68, 1999.
29. H.D. Sherali and G. Choi. Recovery of primal solutions when using subgradient optimization methods to solve Lagrangian duals of linear programs. *Operations Research Letters*, 19:105–113, 1996.
30. M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35:254–365, 1987.
31. R. Sridharan. A Lagrangian heuristic for the capacitated plant location problem with single source constraints. *European Journal of Operational Research*, 66:305–312, 1991.

32. M.G.C. Van Krieken, H. Fleuren, and R. Peeters. A Lagrangean relaxation based algorithm for solving set partitioning problems. Technical Report 2004-44, CentER Discussion Paper, 2004.

Chapter 6
Convergence Analysis of Metaheuristics

Walter J. Gutjahr

Abstract In this tutorial, an overview on the basic techniques for proving convergence of metaheuristics to optimal (or sufficiently good) solutions is given. The presentation is kept as independent of special metaheuristic fields as possible by the introduction of a generic metaheuristic algorithm. Different types of convergence of random variables are discussed, and two specific features of the search process to which the notion "convergence" may refer, the "best-so-far solution" and the "model", are distinguished. Some core proof ideas as applied in the literature are outlined. We also deal with extensions of metaheuristic algorithms to stochastic combinatorial optimization, where convergence is an especially relevant issue. Finally, the important aspect of convergence speed is addressed by a recapitulation of some methods for analytically estimating the expected runtime until solutions of sufficient quality are detected.

6.1 Introduction

Metaheuristic algorithms (see, e.g., the standard textbook [19]) are general-purpose solvers for optimization problems. Their use is prevalent in modern applications of computational intelligence because of their broad flexibility, their intuitive structures and their applicability even in the case of problem instances for which classical optimization techniques fail. A huge amount of knowledge about the *empirical* performance of diverse metaheuristics on numerous classes of optimization problems has been accumulated within the last two decades. Theoretical analysis lags behind this development. While some aspects of the performance of metaheuristics are already well-understood,

Department of Statistics and Decision Support Systems, University of Vienna, Vienna, Austria
e-mail: walter.gutjahr@univie.ac.at

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_6,

some other issues still wait for an analytical treatment. One of the most basic issues is the question whether or not the current solutions proposed by a metaheuristic converge to an optimal or at least to a "sufficiently good" solution—and if yes, how fast this happens.

Especially the recent attempts of combining metaheuristic techniques with the well-established methods from mathematical programming (MP) to powerful hybrids give the convergence issue a new relevance. Researchers from the MP field are used to be sure that their algorithms reach an optimal solution within finite time. More than that, even a proof of optimality is delivered by an MP method, typically within a computation time that can be bounded from above (although the bound may be very large). In contrast to that, metaheuristics are not only unable to produce proven optimality; some often applied variants cannot even guarantee that an optimal solution (or a solution of a certain minimum quality) will be eventually found if an unlimited amount of computation time is invested. An algorithm embedding an MP component within a metaheuristic inherits this weakness.

Fortunately, however, convergence to the optimum *can* be shown for several variants (or parameterizations) of metaheuristic algorithms, and usually, convergence of an algorithm can even be ensured simultaneously for a very broad range of problems, e.g., for all combinatorial optimization (CO) problems. Identifying conditions that entail convergence provides a starting point also for the practically very relevant question of convergence speed.

In this tutorial, we give an introduction to the basic ideas of convergence proofs for metaheuristics. The scope will be restricted by two limitations: First, we shall focus on *global* convergence instead of converge to local optima. In particular, dynamical-systems approaches to the analysis of metaheuristics (cf. [20] or [60, 9]) are not addressed here because of their orientation towards local convergence. Secondly, we concentrate on discrete finite search spaces, i.e., on the CO case. There is a large amount of literature on convergence of search algorithms on continuous search spaces, let us only refer to [56, 59, 45] for some recent examples. Because of the specific aspects occurring in this scenario, however, we must consider a discussion of this type of investigations as outside of the scope of this article.

Since we do not restrict ourselves to some specific metaheuristic algorithm but would rather like to discuss the topic in a general context, we have to start with a generic framework encompassing most (or all) existing metaheuristics. This is done in Section 6.2. Section 6.3 addresses convergence definitions and different types of convergence of metaheuristic algorithms. In Section 6.4, core ideas of convergence proofs are outlined and illustrated by examples concerning different metaheuristics. Section 6.5 deals with convergence of metaheuristic variants for Stochastic CO problems, and Section 6.6 shortly discusses the issue of convergence speed. The final Section 6.7 contains concluding remarks and open research topics.

6.2 A Generic Metaheuristic Algorithm

We consider optimization problems of the form

$$f(x) \to \min \text{ such that } x \in S, \tag{6.1}$$

where S is a search space and f is a real-valued function called *objective function* or *cost function*. Maximization problems are reformulated as minimization problems in a straightforward way. We shall assume S to be a finite set, which puts the problem (6.1) into the field of CO.

To give a unified framework within which convergence may be discussed for a large class of metaheuristics, we present in the following a generic algorithm that includes most (perhaps all) currently used metaheuristic algorithms as special cases (cf. also [28]). In some sense, the proposed generic algorithm extends the generic black-box optimizer presented by Droste et al. [13], but it is not restricted to black-box optimization, and it does not consider prior function samples in their raw form, but rather in an already condensed form, which is closer to existing metaheuristics and automatically addresses space restrictions.

The algorithm (say, in a concrete instantiation A) works in an iterative way. In iteration t ($t \geq 1$), algorithm A uses a memory m_t and a list L_t of solutions $x_i \in S$. We call the elements of L_t *sample points*. The structure of the procedure is the following:

1. Initialize m_1 according to some rule.
2. In iteration $t = 1, 2, \ldots$, until some stopping criterion is satisfied,
   a. determine the list L_t as a function $g(m_t, \xi_t)$ of the memory state m_t and of a random influence ξ_t;
   b. determine the objective function values $f(x_i)$ of all $x_i \in L_t$, and form a list L_t^+ containing the pairs $(x_i, f(x_i))$;
   c. determine the new memory state m_{t+1} as a function $h(m_t, L_t^+, \xi_t')$ of the current memory state m_t, of the list of solution-value pairs L_t^+, and of a random influence ξ_t'.

The memory state m_t can be decomposed as $m_t = (m_t^s, m_t^r)$ into two components m_t^s and m_t^r, where m_t^s, the *sample-generating part*, contains all information in m_t that is actually used by the function g for generating the list L_t of sample points, and m_t^r, the *reporting part*, contains that information that is not used for this purpose. The function h is allowed to use both parts m_t^s and m_t^r for updating the memory. Typically[1], the reporting part contains at least the *best-so-far* solution x_t^{bsf} which is initialized arbitrarily for $t = 1$, and set to x_i each time when in some iteration t, some $x_i \in L_t$ is evaluated to an objective function value $f(x_i)$ better than $f(x_t^{bsf})$. The best-so-far solution

[1] An exception is the context of optimization under noise discussed in Section 6.5, where the best-so-far solution cannot be determined with certainty.

is used as the currently proposed approximation to the optimal solution in iteration t.

The elements ξ_t and ξ'_t can be imagined as vectors of (pseudo-)random numbers that are used by the metaheuristic. Thus, the function $g(m_t, \xi_t)$ specifies, to given memory m_t, a probability distribution for the list of new sample points, whereas the function $h(m_t, L_t^+, \xi'_t)$ specifies, to given memory m_t and current list L_t^+ of solution-value pairs, a probability distribution for the new state of the memory. If the functions g and h are independent of ξ_t resp. ξ'_t, we obtain the special case of a *deterministic* search algorithm. Most metaheuristics, however, are stochastic search algorithms.

Contrary to [13], in our formalism, the functions g and h may use any information on the problem instance. (In order not to overload notation, the problem instance is not written explicitly as an additional argument of g and h.) The important special case where g and h are only allowed to use the knowledge of the search space S and of the problem type, but not of the actual problem instance, is denoted as *black-box optimization*. Some variants of metaheuristics are black-box optimization algorithms, some are not. In particular, algorithms combining a metaheuristic with mathematical programming (see Section 6.1) are typical cases of algorithms that are *not* of black-box type.

The states (m_t, L_t^+) generated during the execution of the algorithm form a *Markov process* in discrete time, since the distribution of the next state (m_{t+1}, L_{t+1}^+) only depends on the current state (m_t, L_t^+). For fixed objective function f, we can consider already the sequence (m_t) $(t = 1, 2, \ldots)$ of memory states as a Markov process, since (via L_t^+, which results from m_t) the distribution of m_{t+1} only depends on m_t.

To illustrate that the generic algorithm above is able to cover well-known metaheuristics as special cases, let us look at some examples:

Generalized Hillclimbing Algorithms: This class of metaheuristics, which we shortly denote as GHCs (generalized hillclimbers), was introduced by Jacobson et al. [43] and contains the well-known *Simulated Annealing* (SA) algorithm and some other stochastic local search algorithms as a special cases. Here, a neighborhood structure $\mathcal{N}$ assigning to each $x \in S$ a set $\mathcal{N}(x) \subseteq S$ of *neighbor solutions* is used. The sample-generating part m_t^s of the memory consists of a single element, the current *search point* x_t. The reporting part m_t^r contains x_t^{bsf} as well as the current iteration counter t. The list L_t of sample points consists of a single element, the currently investigated neighbor solution $y \in \mathcal{N}(x)$ to x.

- To determine L_t from m_t^s, choose a random neighbor y to the element x in m_t^s.
- To update m_t to m_{t+1}, decide by the following stochastic acceptance rule whether y is accepted or not: Draw a random variable R from a distribution $\mathcal{D}_t$ depending on t. If $R \geq f(y) - f(x)$, accept y, i.e., set $m_{t+1}^s = \{y\}$. If

$R < f(y) - f(x)$, reject y, i.e., set $m_{t+1}^s = \{x\}$. The new reporting part m_{t+1}^r is obtained by updating x_t^{bsf} to x_{t+1}^{bsf} and by incrementing t.

SA is the special case where the random variable R is chosen as $-c_t \ln(\zeta)$, where c_t is the *temperature* parameter of SA in iteration t, and ζ is a random number distributed uniformly on $[0, 1]$. The acceptance probability of worse neighbors results then as $\exp(-(f(y) - f(x))/c_t)$.

In other stochastic local search algorithms such as Iterated Local Search (ILS) or the Variable Neighborhood Search (VNS) algorithm developed by Hansen and Mladenović [33], the memory is slightly extended, e.g., by the addition of an *incumbent* solution. Contrary to GHCs and to ILS, VNS works with a family of neighborhoods $\mathcal{N}_k$ of varying sizes $k = 1, \ldots, k_{max}$. In all these metaheuristics, the number of elements in the memory is fixed (and usually small) and is not tuned to the problem instance. This is different in the next example:

Genetic Algorithms (GAs): For the *canonical* GA according to the definition in [54], m_t^s consists of a current population of k solutions, m_t^r contains only x_t^{bsf}, and also L_t consists of k solutions.

- To determine L_t from m_t^s, apply the well-known genetic operators *mutation* and *crossover* to the solutions in m_t. This yields L_t.
- To update m_t to m_{t+1}, apply fitness-proportional *selection* to the population contained in L_t by using the corresponding objective function values. The result is stored as m_{t+1}^s. The new reporting part is obtained by updating x_t^{bsf} to x_{t+1}^{bsf}.

Other GAs are represented as obvious modifications of this procedure.

Ant Colony Optimization (ACO): Both the *Ant System* (AS) variant by Dorigo et al. [10] and the *Max-Min Ant System* (MMAS) variant by Stützle and Hoos [58] can be represented in the following form: The sample-generating part m_t^s of the memory consists of a vector (or matrix) of real-valued parameters, called *pheromone* values. The reporting part m_t^r contains x_t^{bsf} and (depending on the special MMAS version) possibly also an *iteration-best* solution x_t^{ib}. The list L_t consists of k solutions.

- To determine L_t from m_t^s, let k "ants" construct k random solutions by traversing random paths in a "construction graph", where the probabilities of the moves are governed by the pheromone values in m_t^s.
- To update m_t to m_{t+1}, start by updating x_t^{bsf} and x_t^{ib} based on the k new solutions in L_t^+. Then, apply the specific *pheromone update rule* of the ACO variant under consideration in order to determine the new pheromone values from the current values by reinforcing the components of the solution(s) contained in x_t^{bsf} and/or x_t^{ib}. This gives m_{t+1}^s.

The macro-structure of *Estimation-of-Distribution Algorithms* (EDAs, see, e.g., Gonzalez et al. [20]) is very similar to that of ACO. The parameters of

the distribution used for sampling replace here the pheromone values of ACO. Also the *Cross Entropy Optimization* metaheuristic introduced by Rubinstein [53] shows the same macro-structure.

Finally, let us mention that also metaheuristics as *Particle Swarm Optimization* (PSO) (developed by Kennedy and Eberhart [47]) fit into the framework above, although in the standard version of PSO, search is not performed on a discrete finite search space, but on a continuous search space instead. PSO versions for CO problems as the Binary PSO algorithm proposed in [48] have a structure related to that just described for ACO. Here, m_t^s contains real-valued positions and velocities of a set of particles.

6.3 Convergence

6.3.1 Convergence Notions

The mathematical definition of convergence of a series $(x_1, x_2, \ldots)$ of elements in a space $\mathcal{X}$ with a distance function d is the following: The series (x_n) converges to a limit x^*, if for each $\epsilon > 0$, there is an integer N such that $d(x_n, x^*) < \epsilon$ for all $n \geq N$. This definition considerably simplifies if the space $\mathcal{X}$ is finite. In this case, x_n converges to x^* if and only if there is some N such that $x_n = x^*$ for all $n \geq N$. It should be mentioned, however, that although we restrict ourselves to finite *search spaces* in this paper, we shall also have to do with convergence of non-discrete random variables.

Most metaheuristics are *stochastic* search algorithms, such that the definition above is not sufficient. We need a generalization to a notion of convergence of a series of *random variables*. Let us start with a recapitulation of two important established definitions of *stochastic* convergence. After that, we proceed to a consideration of different types of convergence of a metaheuristic.

We consider a stochastic process $(X_1, X_2, \ldots)$, i.e., a sequence of random variables with a common distribution. In general, the random variables are not independent.

Definition 1.

(i) A sequence of random variables $(X_1, X_2, \ldots)$ converges *with probability one* (short: w. pr. 1) or *almost surely*[2] to a random variable X^*, if

$$Pr\{X_t \to X^*\} = 1, \tag{6.2}$$

[2] The last term is very usual in the probabilistic literature. We avoid it in this paper because the word "almost" can be misunderstood. If $X_t \to X$ w. pr. 1, it's quite sure that the sequence converges.

i.e., if with probability one, the realization $(x_1, x_2, \ldots)$ of the sequence (X_t) converges to the realization x^* of X^*.

(ii) A sequence of random variables $(X_1, X_2, \ldots)$ converges *in probability* to a random variable X^*, if for all $\epsilon > 0$,

$$Pr\{\, d(X_t, X^*) \geq \epsilon\} \;\rightarrow\; 0 \text{ as } t \rightarrow \infty, \tag{6.3}$$

where d is the distance function on the space $\mathcal{X}$ in which the random variables X_t take their values.

It can be shown that convergence notion (i) is stronger than convergence notion (ii): if $X_t \rightarrow X^*$ w. pr. 1, then also $X_t \rightarrow X^*$ in probability. In general, the converse does not hold. Let us construct, e.g., a sequence $X_1, X_2, \ldots$ of binary random variables as follows. Initialize each X_t by the value 0. Decompose the index set $\{1, 2, \ldots\}$ into blocks $k = 1, 2, \ldots$ of increasing length k, such that the first block is $\{1\}$, the second block is $\{2, 3\}$, the third block is $\{4, 5, 6\}$, etc. Now select within each block k a position t uniformly at random and set the corresponding variable X_t to the value 1. Then, $X_t \rightarrow 0$ in probability as $t \rightarrow \infty$, but $X_t \rightarrow 0$ w. pr. 1 does not hold, since a realization $(x_1, x_2, \ldots)$ of this stochastic process never converges.

Usually, these definitions are specialized to the case where the limiting X^* is a constant, deterministic element x^*. If $\mathcal{X}$ is a finite set, convergence of X_t to x^* w. pr. 1 holds exactly if

$$Pr\{\text{there is a } u \geq 1 \text{ such that } X_t = x^* \text{ for all } t \geq u\} \;=\; 1,$$

and convergence of X_t in probability holds exactly if $Pr\{\, X_t = x^*\} \rightarrow 1$ as $t \rightarrow \infty$. In the finite case, we can also slightly generalize the definition of convergence in probability by saying that X_t converges to a subset $\mathcal{X}^*$ of $\mathcal{X}$ in probability, if $Pr\{\, X_t \in \mathcal{X}^*\} \rightarrow 1$ as $t \rightarrow \infty$.

We shall apply these definitions to components of the current state (m_t, L_t^+) of the Markov process associated with algorithm A. However, which components should be considered, and how is the limiting element supposed to look like?

6.3.2 Best-So-Far Convergence

A natural choice consists in considering the best-so-far solution x_t^{bsf} and to ask whether or not it converges to some *optimal solution*, as $t \rightarrow \infty$. In a finite search space and with x_t^{bsf} defined as in Section 6.2, convergence of x_t^{bsf} to an optimal solution amounts to convergence of the cost function values $f(x_t^{bsf})$ $(t = 1, 2, \ldots)$ to the optimal cost function value. Thus, we may ask under which conditions it can be guaranteed that $f(x_t^{bsf})$ converges ("w. pr. 1" or "in probability") to $f^* = \min\{f(x) : x \in S\}$.

Restricting the discussion to the case of finite S, we can simplify the defining conditions (6.2) – (6.3) for convergence of $f(x_t^{bsf})$ to the optimum f^* by introducing the nondecreasing sequence of indicator variables

$$Z_t = I(f(x_t^{bsf}) = f^*) \quad (t = 1, 2, \ldots),$$

where I denotes the indicator function. The *success indicator* Z_t is 1 if an optimal solution is found in one of the iterations $1, \ldots, t$, and 0 otherwise. The *first hitting time* is given as

$$T_1 = \min\{t \geq 1 : \ Z_t = 1\}. \tag{6.4}$$

Furthermore, with E denoting the mathematical expectation, we define

$$\mu_t = E(Z_t) = Pr\{Z_t = 1\} = Pr\{T_1 \leq t\} \quad (t = 1, 2, \ldots) \tag{6.5}$$

as the probability that algorithm A finds an optimal solution in one of the iterations $1, \ldots, t$. (In some articles such as [35] or [64], the sequence of numbers $1 - \mu_t$ is called the *convergence rate*.) It is easy to see that with this notation and with the ordinary absolute difference on the set of reals as the distance function,

- $f(x_t^{bsf})$ converges to f^* w. pr. 1 if and only if $Pr\{Z_t = 0 \text{ for all } t\} = 0$ (which is the same as $Pr\{T_1 < \infty\} = 1$), and
- $f(x_t^{bsf})$ converges to f^* in probability if and only if $\mu_t \to 1$ $(t \to \infty)$.

It follows that for best-so-far convergence, the two convergence notions coincide, since $Pr\{Z_t = 0 \text{ for all } t\} \leq Pr\{Z_u = 0\} = 1 - \mu_u$ for all iterations u, such that $\mu_u \to 1$ as $u \to \infty$ implies convergence w. pr. 1.

The convergence concept above may look nice. However, it has a serious disadvantage: It turns out that under this concept, even very inefficient search algorithms converge to the optimum, which makes the concept too "generous". The standard example for this observation is random search.

In our framework, (pure) random search can be described as that instantiation of our generic algorithm where the sample-generating part m_t^s of the memory is empty, the reporting part m_t^r consists only of x_t^{bsf}, and the list L_t consists of a single sample point x_t that is chosen at random from S according to some fixed distribution. Because m_t^s does not contain any information, the choice of x_t has to be performed *independently* of the memory state m_t (and hence of the previous iterations).

It is well-known that random search on a finite set S ensures convergence of the best-so-far solution value to the optimum, as long as every solution $x \in S$ has a nonzero probability $p(x) > 0$ of being chosen as the sample point x_t. We will derive this quickly in Subsection 6.4.1. Furthermore, however, it will be shown that the expected value $E(T_1)$ of the first hitting time is usually very large for random search, such that convergence is here of no help for practice.

Informally, the reason why convergence in the best-so-far sense does not go hand in hand with a good runtime behavior is that for demonstrating this type of convergence, it is only required that the algorithm performs a sufficient amount of *exploration* of the search space. The feature of *exploitation* of the information obtained in previous iterations (stored in the memory m_t), as it is incorporated in most well-performing metaheuristics, does not alleviate the convergence proof; quite contrary, it rather hampers it. Of course, however, exploitation is an advantage for the runtime of the algorithm! Thus, convergence of the best-so-far solution value is not an indicator for a good runtime behavior. It only ensures that the part of the search space containing the optimal solution is not excluded from the search *a priori*, such that at least in the absence of limits on computation time, search is "complete".[3]

Several investigations on the convergence of diverse metaheuristics have started with results concerning the best-so-far concept. Let us give examples from two metaheuristic fields: Hartl [34] and Rudolph [54] showed convergence of $f(x_t^{bsf})$ to the optimum for certain variants of GAs transferring "elite" solutions (best solutions in a current population) from generation to generation. If, e.g., one elite solution is always preserved within the population, this solution is identical to x_t^{bsf}. Brimberg et al. [7] showed convergence results of best-so-far type for ILS as well as for a VNS parametrization where the parameter k_{max} defining the largest neighborhood is sufficiently large to cover the whole search space S. The proofs provided in the mentioned articles are possible starting points for the derivation of stronger convergence properties (cf. the remarks in the next subsection).

6.3.3 Model Convergence

The chance that provable convergence properties are correlated with good runtime behavior are considerably increased if we do not focus on the best-so-far solution x_t^{bsf}, but on the sample-generating part m_t^s of the memory. In an efficient metaheuristic, exploitation of the search experience should concentrate the search more and more on the most promising areas of the search space S, with the consequence that the average cost function values of the sample points in L_t tend to decrease (not necessarily monotonically) over time. The case where the expected cost function values in L_t remain constant is essentially the exploitation-less random search case.

Borrowing from the concept of "model-based search" as developed by Zlochin et al. [65], we may alternatively denote the sample-generating part m_t^s of the memory as the current *model* for the search distribution. In the model-based view, search points are generated in dependence of the model,

[3] Hoos [39] and Hoos and Stützle [40] call a search algorithm with $Pr\{T_1 < \infty\} < 1$ for a class of problems *essentially incomplete* for this class.

cost function values are evaluated, and the obtained information is then fed back into a modification of the model. Basically, this corresponds to the mechanism of our generic algorithm, with the difference that we also extend this view to classical search algorithms with discrete state spaces instead of restricting it to metaheuristics as ACO, EDAs or Cross Entropy Optimization, where the model is described by a vector of *real-valued* parameters.

By the argumentation above, a runtime behavior superior to that of random search can be expected if it can be shown that the *model* m_t^s converges, as $t \to \infty$, to some limiting state $(m^s)^*$ that supports only the generation of optimal or at least high quality sample points. We denote this type of convergence as *model convergence* as opposed to best-so-far convergence. Note that the model can be very small: in a GHC, e.g., it contains only the current search point x_t. Nevertheless, convergence of this search point to a solution in S^* is more relevant than convergence of x_t^{bsf} resp. $f(x_t^{bsf})$ only, since it indicates that the search is gradually directed towards more promising areas of the search space.[4]

Contrary to proofs of best-so-far convergence which are technically the easier the more emphasis the considered algorithm puts on exploration (as opposed to exploitation), model convergence proofs have to take the *exploration-exploitation tradeoff* explicitly into account and only succeed under parameter assumptions ensuring a proper balance between these two factors. Typically, the results yield rather narrow conditions for parameter schemes within which model convergence holds; outside the balanced regime, either a surplus of exploitation yields premature convergence to a suboptimal solution, or a surplus of exploration produces random-search-type behavior without model convergence (although best-so-far convergence may hold).

Historically, the first model convergence results have been found in the SA field (see, e.g., Gelfand and Mitter [17], Hajek [32], or Aarts and Korst [1]). In Subsection 6.4.2, we shall outline the key ideas of the proofs in the more general context of modern GHC results. Also for some ACO variants and for a variant of Cross Entropy Optimization, results of model convergence type are known. Empirical evidence suggests that such results should be possible for several other metaheuristics as well.

In the GA case, model convergence would mean that the population tends to "positive stagnation" in the long run by being filled with optimal solutions only. Since mutation counteracts this effect, a model-convergent variant of a GA would presumably have to gradually decrease the mutation rate and/or to increase the selection pressure. This would have to be done slowly enough

[4] Convergence analysis of metaheuristics has been questioned by the argument that every randomized search algorithm can easily be made convergent to the optimum by repeatedly calling it with randomly chosen start solutions. This is trivial for best-so-far convergence, as already the series of start solutions yields a random search run in itself. However, *model* convergence typically does *not* hold in this scenario, since during each restart, the current information in m_t^s is thrown away. Thus, the expected average cost function value in the sample points does not improve from run to run.

to prevent premature convergence and fast enough to ensure convergence. Similarly investigation could be performed for the VNS case. (Convergence of the Markov process of VNS visits in local optima has been shown in Brimberg et al. [7].)

6.4 Proving Convergence

6.4.1 Proving Best-So-Far Convergence

Convergence of x_t^{bsf} to the optimum can usually be shown easily. We shall illustrate this for the simple case of random search. Let Z_t be defined as in Section 6.3, and let $p = \sum_{x \in S^*} p(x) > 0$ denote the probability of hitting an optimal solution in a single iteration of the random search algorithm. Because of the independence of the trials in different iterations, $Pr\{Z_t = 0\} = (1-p)^t \to 0$ as $t \to \infty$, and therefore $\mu_t \to 1$ as $t \to \infty$. In other words, convergence in probability to the optimum holds (and by the remark in Subsection 6.3.2 on the special situation for best-so-far convergence, even convergence w. pr. 1).

In some more interesting algorithms, the hitting probability is time-dependent: $p = p_t$. If a strictly positive lower bound $p_{min} > 0$ for p_t can be shown, convergence immediately results as above. However, even in some cases where p_t tends to 0, convergence still holds: E.g., if $p_t = ct^{-1/2}$ $(c > 0)$, we still get convergence in probability since $\lim_{t\to\infty}(1 - ct^{-1/2})^t = 0$. For $p_t = c/t$, this does not hold anymore, since $(1 - c/t)^t \to e^{-c} > 0$. Lower bounds on the hitting probability have been used, e.g., in the convergence analysis of the MMAS variant of ACO given by Stützle and Dorigo [57].

For time-independent p, the expected value $E(T_1)$ of the first hitting time computes as $1/p$ by the standard calculation of the expected value of a geometric distribution, which is typically a very large number even for medium-sized search spaces, unless if the distribution $p(\cdot)$ of the random trials can be focused around the set S^* of optimal solutions by means of prior information.

6.4.2 Proving Model Convergence

We outline the ideas of model convergence proofs by presenting some characteristic examples.

6.4.2.1 Generalized Hillclimbers and Simulated Annealing

The results by Jacobson and Yücesan [44] concerning convergence of GHCs are especially instructive as, on the one hand, GHCs contain SA (for which the first model convergence theorems have been shown) as a special case, and on the other hand, the article works already on a more general level such that metaheuristic-independent features become visible. In [44], it is assumed that subsequent iterations of a GHC are comprised to *macro iterations* $k = 1, 2, \ldots$. (We shall choose a simple, special way of defining macro iterations later.) During each macro iteration, the random variable R deciding on acceptance (see Section 6.2) has the same distribution. With $t(k)$ denoting the last "micro" iteration of macro iteration k, let x^k denote $x_{t(k)}$, i.e., the current search point as it is obtained at the end of macro iteration k. We introduce the following abbreviations:

- $C(k)$ is the event that $x^k \in S^*$, i.e., the event that macro iteration k produces an optimal solution. The complementary event is denoted by $C^c(k)$.
- $B(k)$ is the event $C^c(1) \cap C^c(2) \cap \ldots \cap C^c(k)$, i.e., the event that none of the macro iterations $1, \ldots, k$ produces an optimal solution. The complementary event to $B(k)$ is $B^c(k)$.
- $B = \bigcap_{k=1}^{\infty} B(k)$ is the event that no iteration at all produces an optimal solution.
- $r(k) = Pr\{B^c(k) \,|\, B(k-1)\}$ is the probability that in macro iteration k, an optimal solution is produced, although it has not yet been produced in any of the previous macro iterations.

Convergence of x^k in probability to the set X^* of optimal solutions can be expressed as $Pr(C(k)) \to 1$ as $k \to \infty$. It is now possible to show the following criterion:

Theorem 1 (Jacobson and Yücesan [44]). For a GHC, x^* converges to S^* in probability if and only if the two following two conditions are satisfied:

(i) $\sum_{k=1}^{\infty} r(k) = \infty$,
(ii) $\Pr(C^c(k) \,|\, B^c(k-1)\} \to 0$ as $k \to \infty$

Let us present the proof idea of the part of the theorem stating that the two conditions above are *sufficient* for convergence in probability. The idea is not too complicated, but very informative, because it recurs in some variations in related proofs in the literature. First, we show that condition (i) is equivalent to $Pr(B) = 0$. This results as follows:

$$Pr(B) = Pr(B(1)) \cdot Pr(B(2)|B(1)) \cdot Pr(B(3)|B(1) \cap B(2)) \cdot \ldots$$

$$= (1 - r(1)) \cdot (1 - r(2)) \cdot (1 - r(3)) \cdot \ldots .$$

Therefore,

$$Pr(B) = 0 \Leftrightarrow \prod_{k=1}^{\infty}(1 - r(k)) = 0 \Leftrightarrow \sum_{k=1}^{\infty}\log(1 - r(k)) = -\infty,$$

where the second equivalence follows by taking logarithm on both sides. Since $\log(1 - r(k)) \sim -r(k)$ for small $r(k)$, the latter is equivalent to $\sum_{k=1}^{\infty} r(k) = \infty$. (To make the proof precise, the approximation has to be replaced by bounds.)

Now, by the law of total probability,

$$\begin{aligned} Pr(C^c(k)) &= Pr(C^c(k)|B^c(k-1)) \cdot Pr(B^c(k-1)) \\ &\quad + Pr(C^c(k)|B(k-1)) \cdot Pr(B(k-1)) \\ &= Pr(C^c(k)|B^c(k-1)) \cdot Pr(B^c(k-1)) + P(B(k)). \end{aligned}$$

By condition (ii), the first term in the last expression tends to 0 as $k \to \infty$. Because of the equivalence derived above, condition (i) yields $Pr(B) = 0$. Because $B(1) \subseteq B(2) \subseteq \ldots$, by the Monotone Convergence Theorem,

$$Pr(B(k)) \to Pr\left(\bigcap_{k=1}^{\infty} B(k)\right) = Pr(B) = 0,$$

and therefore also the second term tends to zero. This shows $Pr(C^c(k)) \to 0$, which completes the proof.

Theorem 1 can be nicely interpreted in terms of the exploration-exploitation tradeoff: Condition (i) guarantees that enough exploration is performed in order to be sure to find a globally optimal solution eventually. Condition (ii), on the other hand, ensures that enough exploitation is done in order to preserve an optimal solution with a high probability, once it has been found, and enables convergence in this way.[5]

Consider now the special case of SA. We choose macro iterations of equal length, consisting of L micro iterations each, where L is the maximum of the minimum number of transitions to neighbors required to reach an optimal solution from an arbitrary initial solution x over all $x \in S$. Then it is possible to reach from an arbitrary point $x \in S$ an optimal solution in exactly L micro

[5] The decomposition of the process into a phase before and a phase after an optimal solution x^* has been found may appear as a cheap trick: One might be tempted to construct a "model-convergent" metaheuristic by letting it perform random search before x^* has been found, and to freeze the current solutions x_t to x^* after that time. The point is, however, that the decomposition into these two phases only exists at the level of analysis and cannot be done by the algorithm itself, which does not know when it has detected an optimal solution, such that it cannot use the attainment of the optimum as the criterion for switching from exploration to exploitation. Thus, in order to preserve x^* after it has been discovered, the algorithm has to do a sufficient amount of exploitation already before this event — which, on the other hand, makes it nontrivial to guarantee that the global optimum is not missed.

iterations (i.e., in one macro iteration), either by moving with the search point x_t towards x^* or by letting it stay in x^*, rejecting neighbor solutions. We shall focus on the process (x^k) defined by the macro iterations, but mention that corresponding results can also be derived for the process (x_t) on the level of micro iterations.

It is easy to see that if the temperature parameter is fixed at some constant level c, either condition (i) or condition (ii) of Theorem 1 are violated: If $c > 0$, then $r(k)$ has a strictly positive lower bound, such that condition (i) holds; in this case, however, condition (ii) is not satisfied, since even after an optimal solution has been visited, suboptimal solutions will always be produced with probabilities larger than some positive constant. On the other hand, if $c = 0$, then only better neighbor solutions are accepted, with the consequence that condition (ii) is satisfied (once an optimal solution has been found, it is not left anymore), but condition (i) is violated, because usually an optimal solution is not found.

The stunt of satisfying the two conditions simultaneously is achieved by decreasing the temperature parameter with a suitable speed. Choose a temperature scheme c_k with $c_k \to 0$ $(k \to \infty)$ and $c_k \geq L\Delta/\log(k+1)$ $(k = 1, 2, \ldots)$, where $\Delta = \max_{x,y \in S}(f(x) - f(y))$. It is easily seen that as soon as the temperature has become low enough,

$$r(k) \geq \left[\frac{1}{|S|} \cdot \exp\left(-\frac{\Delta}{c_k}\right)\right]^L \geq \frac{C}{k}$$

with some constant $C > 0$, which implies that $\sum r(k) = \infty$ and hence condition (i) of Theorem 1 is satisfied.

To show that the above temperature scheme is also sufficient for satisfying condition (ii) needs some technicalities from the theory of inhomogeneous Markov chains, which we omit here; the interested reader is referred to [1]. Let us only provide the rough picture. The sequence (x_t) is an inhomogeneous Markov chain with transition matrix $P(k)$ on temperature level c_k. Condition (i) above ensures *weak ergodicity* of this Markov chain, which essentially means that the dependence on the initial solution vanishes over time. To satisfy also condition (ii), it has to be shown that (a) for all k, there exists a left eigenvector $\pi(k)$ of $P(k)$ with eigenvalue 1, (b) the eigenvectors $\pi(k)$ satisfy $\sum_{k=1}^{\infty} ||\pi(k) - \pi(k+1)|| < \infty$, and (c) the eigenvectors π_k converge as $k \to \infty$ to a limiting vector π^* containing probabilities of the solutions $x \in S$ such that only the probabilities in S^* have nonzero values. These properties can be demonstrated to be satisfied indeed for the given temperature scheme, which implies that the Markov chain is strongly ergodic and converges in distribution to π^*.

6.4.2.2 Ant Colony Optimization and Cross Entropy Optimization

The convergence proofs for special ACO variants in [21, 22] and for a particular variant of Cross Entropy Optimization in [49] have a similar structure as the results for GHCs outlined above. Let us explain this for the ACO case.

The algorithmic variants investigated in [22] are derived from the MMAS variant of ACO proposed in [58]. MMAS provides the possibility of applying a *lower pheromone bound* $\tau^{min} > 0$ which prevents that the components of the pheromone vector τ_t contained in m_t^s approach zero. In this way, exploitation is limited to a certain degree in favor of exploration. The update of the pheromone values is done by a reinforcement of the components of the best-so-far solution x_t^{bsf}, sometimes also the components of the iteration-best solution x_t^{ib} are reinforced. The degree of reinforcement is controlled by a parameter $\rho \in]0, 1[$ called *evaporation rate*, which can be considered as a learning rate. The new pheromone vector results as

$$\tau_{t+1} = (1 - \rho)\tau_t + \rho\psi_t,$$

where ψ_t is the vector of the rewards in the current iteration. If ρ is high, the observations made in the current iteration (the "presence") have a high influence on the new pheromone vector, compared to former iterations (the "past"), whereas in the more conservative case of low ρ, the past is given a higher influence than the presence.

In [22], two particular variants are considered: The first of them, "algorithm 1", does not use a pheromone bound, but decreases the learning rate ρ over time, choosing it as $\rho = \rho_t$. The second one, "algorithm 2", uses a time-dependent lower pheromone bound τ_t^{min}. In both algorithms, iteration-best reinforcement is not done, i.e., x_t^{ib} is not used.

The following result is shown: Both for algorithm 1 and for algorithm 2, as $t \to \infty$, x_t^{bsf} converges w. pr. 1 to an optimal solution x^*, *and* the current state τ_t of the sample-generating part m_t^s of the memory converges w. pr. 1 to a pheromone vector τ^* allowing only the generation of the optimal solution x^*, provided that the following conditions are satisfied:

- In the case of algorithm 1:

$$\rho_t \leq 1 - \frac{\log t}{\log(t+1)} \quad \text{and} \quad \sum_{t=1}^{\infty} \rho_t = \infty.$$

  This can be achieved, e.g., by the parameter scheme $\rho_t = c/(t \log t)$ with $0 < c < 1$.
- In the case of algorithm 2:

$$\tau_t^{min} = c_t / \log(t+1) \quad \text{with} \quad \lim_{t\to\infty} c_t > 0.$$

  This can be achieved, e.g., by constant $c_t = c > 0$.

The line of the proof is similar to the general scheme implicit in the proof of Theorem 1. Also here, the first part of the proof consists in ensuring that eventually, an optimal solution x^* is found. For this purpose, it is demonstrated that from the indicated conditions, a counterpart to condition (i) in Theorem 1 follows. After that, it has to be ensured that w. pr. 1, pheromone concentrates on the components of x^* and vanishes elsewhere. (That the optimal solution x^* is not left anymore after the first visit is trivial here by construction of x_t^{bsf}.) The convergence of the pheromone vector results by a deterministic consideration for an arbitrary realization of the stochastic process, such that we even obtain convergence w. pr. 1 in this scenario. Note that the conditions for algorithm 1 require that ρ_t decreases neither too fast nor too slow, and the conditions for algorithm 2 require an analogous property for the lower pheromone bounds τ_t^{min}.

Whereas in [22] it was assumed that the rewards for the components of the best-so-far solution are of a constant amount, Sebastiani and Torrisi [55] gave convergence conditions for the modification of the MMAS algorithm where the size of the rewards is chosen "fitness-proportional", i.e., decreasing in the cost function value of the solution to be reinforced. (This modification is frequently used in practice.)

Margolin [49] was able to provide a convergence proof for a certain Cross Entropy Optimization variant. Both convergence conditions and proof technique are closely related to the results in [22], so that we omit the details here.

6.4.2.3 Practical Aspects

What do the outlined results imply for applications of metaheuristics? We do not claim that it is always advisable to use a model-convergent parameter scheme when implementing a metaheuristic. Rather than that, our claim is that it can be helpful to know how such a scheme looks like. Let us justify this by an informal argument: In some sense, a model-convergent parameter scheme is a scheme maximizing exploitation under the constraint that there is still a sufficient amount of exploration in order to keep the promise of finally achieving the globally optimal solution alive. In cases of large problem instances with many good local optima, it may be more efficient to sacrifice the warranty of finding the global optimum "at the end of the day" for the benefit of accelerating the search for good local optima "in the course of the day". In such a case, one may decide to apply a parametrization of the algorithm that is slightly below the model-convergent scheme with respect to the degree of exploration. This means slightly faster cooling in SA or slightly less restrictive lower pheromone bounds in ACO, compared to the conditions in the theoretical convergence results.

Future research, both on a theoretical and on an experimental level, may possibly concretize this consideration by quantitative results.

6.5 Convergence for Problems with Noise

There is a situation where the concept of best-so-far convergence is not applicable at all. It is the scenario of *Stochastic Combinatorial Optimization* characterized by the property that the parameters of either cost function or constraints are not known with certainty, such that stochastic models are required to represent these parameters as random variables. We restrict ourselves to the special (but frequently occurring) case where only the cost function is uncertain, and the objective is to minimize its expected value. If this expected value can be computed efficiently, the situation reduces to that of deterministic CO and can be treated by ordinary metaheuristics. Otherwise, a solution algorithm has to be based on concrete observations (realizations) of the cost function values in certain points $x \in S$, but these observations do not represent the expected value, but deviate from it by "noise". A typical case is that where expected costs are estimated by (Monte Carlo) simulation, as it is frequently done, e.g., in queuing systems or in stochastic vehicle routing.

The SCO problem has then the following general form:

$$E(f(x,\omega)) \rightarrow \min \text{ such that } x \in S. \tag{6.6}$$

Therein, ω denotes a random influence (with a distribution given by the stochastic model of the problem), which has to be distinguished from the random variables ξ and ξ' used by the metaheuristic solution procedure (see Section 6.2), and E is the expectation with respect to the distribution of ω. For surveys on the solution of problems of this type by metaheuristics, see Jin and Branke [46] and Bianchi et al. [3].

A *sample average estimator* (SAE) approximates the true objective function value $F(x) = E(f(x,\omega))$ by the average over a random sample:

$$\tilde{F}(x) \;=\; (1/s) \sum_{\nu=1}^{s} f(x,\omega_\nu) \;\approx E(f(x,\omega)). \tag{6.7}$$

Therein, ω_ν $(\nu = 1,\ldots,s)$ represent s independent sample observations, e.g., s runs of a simulation routine. It is immediately seem that $E(\tilde{F}(x)) = F(x)$, i.e., the SAE is always unbiased. We can apply the SAE during a metaheuristic optimization run every time when an objective function evaluation is required, but we have to keep in mind that $\tilde{F}(x)$ usually deviates from $F(x)$, and this is so even more if the sample size s is small. Increasing the sample size s improves the accuracy of the SAE, but this comes at the price of increasing the runtime. The resulting tradeoff has to be addressed by efficient metaheuristic variants.

Several modifications of different metaheuristics have been proposed in the literature for the treatment of SCO problems, e.g., in the SA field [18, 30, 2], in the ACO field [23, 24, 4] or in the VNS field [29]. These techniques are usually

variable-sample approaches extending the generic algorithm of Section 6.2 in the following way: In step 2b of the generic algorithm, the evaluation of the objective function values $f(x_i)$ is replaced by the determination of an SAE $\tilde{F}(x_i)$. Typically, the sample size $s = s_t$ is increased from iteration to iteration, but there are also variants where s is kept constant.

A crucial observation is that in this scenario, the best-so-far solution x_t^{bsf} cannot be determined anymore, since we cannot decide with certainty which of two solutions x and y has the better objective function value; by increasing s, a guess based on the SAEs $\tilde{F}(x)$ and $\tilde{F}(y)$ becomes more trustable, but never certain. Convergence issues get very important in this framework, because in a non-convergent situation, we would neither know which of the visited solutions to deliver as the proposed solution nor when to stop the algorithm: diminishing marginal gains of $f(x_t^{bsf})$, as they can be used as a stopping criterion in the deterministic context, do not give us a hint here.

The key idea to obtain convergence in the outlined stochastic context is to carefully control the accuracy of the estimates $\tilde{F}(x) = \tilde{F}_t(x)$ as t increases by corresponding increments of the sample size $s = s_t$ in iteration t, such that the influence of randomness is gradually reduced while the metaheuristic under consideration turns to more promising areas of the search space. As a consequence, it becomes more probable in later phases that the global optimum is recognized as such by the search procedure, whereas in earlier phases, where the average solution quality is only small, it would be a waste of time to strive for a very good accuracy in objective function evaluation.

In [30], this idea is carried out by proving that, on some conditions, convergence in probability of SA carries over to the stochastic context. The most essential condition is that the standard deviations of the noise variables $\tilde{F}_t(x) - F(x)$ decrease as $t \to \infty$ with an order $O(t^{-\gamma})$ where $\gamma > 1$. Since the variance of the SAE is inversely proportional to the sample size, this can be achieved by letting the sample size s_t grow faster than quadratically in t.

A related approach has been followed in [23] for proving convergence of a variant S-ACO of ACO proposed for SCO problems. In the design of S-ACO, an attempt has been made to improve performance by considering, instead of the current search point x_t, something that corresponds to the best-so-far solution x_t^{bsf} in the deterministic context: Let $\hat{x}_t^{bsf}$ denote the *presumably* best-so-far solution in iteration t, defined by an arbitrary initialization in iteration 1 and a replacement of $\hat{x}_t^{bsf}$ each time when in some subsequent iteration, the SAE $\tilde{F}_t(x_t)$ of the current solution x_t found in this iteration turns out as better than the SAE $\tilde{F}_t(\hat{x}_t^{bsf})$. In other words, at the end of each iteration t, we perform a *tournament* between the current $\hat{x}_t^{bsf}$ and the current x_t, based on a sample of size s_t, and define the new presumably best-so-far solution as the winner of this tournament. It is shown in [23] that for a modification of the MMAS "algorithm 2" described above with a sample size s_t growing at least linearly in t, convergence of $\hat{x}_t^{bsf}$ to the optimal solution of (6.6) is ensured.

The tournament concept has also been used in [29] to present a provably convergent version S-VNS of VNS for SCO problems. In the proof, a general theorem by Homem-de-Mello [8] is applied, which can possibly also be useful in the convergence analysis of other SCO-metaheuristics. Therefore, let us shortly outline its idea.

Denote the vector of independent random numbers that are used in the tournament of iteration t for the evaluation of $\tilde{F}(\hat{x}_t^{bsf})$ and of $\tilde{F}(x_t)$ (we can take the same vector for both SAEs) by $\omega^t = (\omega_1^t, \ldots, \omega_{s_t}^t)$. The same solution x can take part in the tournament in several iterations, possibly even infinitely often. An interesting question is how fast the sample sizes s_t have to be increased such that we can be sure that after sufficient time, a suboptimal x is distinguished reliably from an optimal x, and (if present) only optimal solutions will win the tournament in the future. Homem de Mellos's theorem, which is proved by large-deviations theory, answers this question:

Theorem 2 (Homem-de-Mello [8], Proposition 3.2). Suppose that for a scheme $(s_1, s_2, \ldots)$ of sample sizes and independent random variables ω_ν^t,

(i) for each $x \in S$, the variances $var[f(x, \omega_1^t)]$ are bounded by some constant $M(x) > 0$,

(ii) the variables ω_ν^t are identically distributed, and the SAEs

$$\tilde{F}_t(x) = (1/s_t) \sum_{\nu=1}^{s_t} f(x, \omega_\nu^t)$$

are unbiased[6], i.e., $E(\tilde{F}_t(x)) = F(x)$ for all x,

(iii) $\sum_{t=1}^{\infty} \alpha^{s_t} < \infty$ for all $\alpha \in]0, 1[$.

Then for each x, we have $\tilde{F}_t(x) \to F(x)$ $(t \to \infty)$ w. pr. 1.

In the S-VNS algorithm [29], a tournament is only performed at the end of each macro iteration, where a macro iteration consists of a shaking step followed by local search. For simplicity, let us apply t as an index for macro iterations in the following. Convergence of S-VNS is shown as follows: First of all, it is demonstrated that with a probability larger than zero, a macro iteration finds an optimal solution x^* and exposes it to the tournament. With probability one, this even happens in infinitely many macro iterations. By using Theorem 2, it is verified that for a specific realization of the stochastic process, among all macro iterations where x^* is exposed to the tournament, there is one (say, macro iteration t^*) from which on the sampling error is already small enough to distinguish reliably between optimal and suboptimal solutions. In t^* and in all subsequent macro iterations, an optimal solution will win the tournament, which proves the assertion.

[6] [8] also refers to a more general situation where $E(f(x, \omega_\nu^t))$ can be different from $F(x)$. In our context, unbiasedness follows by definition.

Condition (iii) of Theorem 1 requires that s_t grows fast enough. It is easy to see that a growth proportional to $t^{1/2}$ already satisfies condition (iii), whereas a growth proportional to $\log t$ is yet too slow. The other conditions of Theorem 2 are usually automatically satisfied.

We see that the proof relies on a convergence property of (only) best-so-far type for the underlying *deterministic* VNS algorithm, which makes the result weaker than those for SA and for ACO. It would be desirable to extend it to a result of model-convergence type.

6.6 Convergence Speed

After having ensured convergence of a metaheuristic algorithm A, the natural next question is "How fast does A converge on a given problem instance?"[7] We have seen in the previous sections that for several metaheuristics, very general convergence results, covering the whole range of CO problems, can be derived. Unfortunately, it is rather unlikely that the next step can be to show comparably broad positive results for the speed of convergence. The reason lies in the so-called *No-Free-Lunch Theorems* (NFLTs) by Wolpert and Macready [63] stating that in the average over *all* cost functions $f : S \to W$, where W is some value range, the expected performance of every black-box optimization algorithm is the same. In particular, in this average, no metaheuristic A is better than random search, and to every function f_1 where A outperforms random search, there must be another function f_2 where random search outperforms A.

The chance to prove a convergence speed faster than that of random search for an algorithm increases for restricted sets of problems or problem instances. NFLTs do not hold within problem classes with restricted computational complexity (see [11, 14, 15, 5]) or for objective functions with certain fitness landscape properties (see [41, 42]). At the moment, however, it has not yet been achieved to derive *general* convergence speed bounds using these observations. Rather than that, the literature on the optimization time of metaheuristic algorithms is split into a large number of single results for special algorithms applied to special (usually rather simple) objective functions. It cannot be the goal of this paper to survey these results. Overviews have recently been given in [51] for evolutionary algorithms with the exception of the swarm-intelligence algorithms ACO and PSO, and in [26] for ACO. However,

[7] It does not make too much sense to raise the second question before the first is answered at least for special problem instances and in the weakest possible meaning. E.g., if for an algorithm A on a given problem instance, x_t^{bsf} does not converge in probability to a set $\tilde{S}$ of solutions considered as sufficiently good (which can be the set S^*), then the expected first hitting time of $\tilde{S}$ is ∞. The concept of convergence or runtime "with overwhelming probability" has been used to deal with situations where non-convergence cannot be excluded, but its interpretations are distinctly less intuitive than those of expected runtimes.

a short recapitulation of some main tools for analyzing convergence speed applicable to *several* metaheuristics may be helpful.

For the sake of brevity, we focus on the possibly most important performance measure, the expected first hitting time $E(T_1)$ of the optimal solution, where T_1 is defined by (6.4). Note that in view of (6.5), the distribution function of T_1 is given by $t \mapsto \mu_t$. If, from the viewpoint of application, it is sufficient to reach a solution of some fixed minimum quality instead of the exact optimum, the concept can easily be modified by considering $E(\tilde{T}_1)$ instead of $E(T_1)$, where $\tilde{T}_1$ is given as $\min\{t \geq 1 : \ f(x_t^{bsf}) \leq c\}$ with some aspiration level c for the cost function.

The following outline of methods is neither intended to be complete nor to present all the formal details. Rather than that, the aim is the presentation of some often applied basic ideas.

(1) Markov Chain Analysis

In the case where the memory content m_t can only take finitely many values, the property that the stochastic process (m_t) is a Markov process can be used directly for computing $E(T_1)$ — at least in principle. In this case, (m_t) is a (homogeneous) *Markov chain*. Examples are GHCs (including SA) and GAs. For the ease of notation, let us assume in the sequel that the optimal solution x^* is unique, and that even the state m^* of the memory producing the optimal solution is unique; this scenario can easily be generalized to the existence of several optimal memory states. We ask for the expected time until state m^* is visited first. Let $P = (p_{ij})$ $(1 \leq i, j \leq N)$ denote the transition matrix of the Markov chain. Arrange the indices of the possible memory states in such a way that the largest index N corresponds to m^*, and let $\hat{P}$ denote the matrix obtained from P by deleting the Nth column and the Nth line. Then it is well-known (see, e.g., Mühlenbein and Zimmermann [50] or He and Yao [37]) that the vector $\vartheta = (\vartheta_1, \ldots, \vartheta_{N-1})$ of the values for $E(T_1)$ after a start in memory state $1, \ldots, N-1$, respectively, computes as

$$\vartheta = (I - \hat{P})^{-1} \cdot 1_{N-1}, \tag{6.8}$$

where I is the identity matrix, and 1_{N-1} is the vector consisting of $N-1$ ones. The inversion of the matrix $I - \hat{P}$ may cause difficulties, but in some special cases, explicit formulas can be derived. E.g., [50] gives a formula for the case where $I - \hat{P}$ has a tri-diagonal form, i.e., contains only three nonzero elements in each line.

(2) State Space Decomposition and Subgoals

Usually, the state space becomes to large to be tractable directly by (6.8). However, in some cases, it can be decomposed into subsets that are treated equivalently by the metaheuristic under consideration. If the transition probability between an element of subset i and an element of subset j only depends on i and j, then the Markov property is also satisfied for the "embedded" chain which has the subsets as its states. This can reduce the size of the state

space considerably. To give a simple concrete example, let us consider the function

$$\sigma_{\alpha,\beta,n}(y) = \begin{cases} y, & \text{if } 0 \le y < \alpha, \\ -y + 2\alpha, & \text{if } \alpha \le y < \beta, \\ y + 2\alpha - 2\beta, & \text{if } \beta \le y \le n \end{cases}$$

on $\{0, \ldots, n\}$, where $0 < \alpha < \beta < n$ and $\beta < 2\alpha$. The function σ has a global minimum at $y = 0$ and a local minimum at $y = \beta$. Furthermore, we define the cost function

$$f_{\alpha,\beta,n}(x) = \sigma_{\alpha,\beta,n}(|x|) \tag{6.9}$$

on the set $S = \{0,1\}^n$ of binary strings of length n, where $|x|$ denotes the number of 1-bits in $x \in \{0,1\}^n$. For minimizing $f(x)$, we apply a GHC where the random variable R takes the value 0 with probability $1 - p$ and the value ∞ with probability p, such that better neighbors are always accepted, whereas worse neighbors are only accepted with probability p. (For our special example, this can also be formulated in terms of SA.) A neighbor solution to x is obtained by flipping a single bit in x. As subset i of the state space, we consider all $x \in S$ with $|x| = i$. Figure 6.1 shows the plot of the expected first hitting time (assuming random initial states) in dependence of the parameter p for the special case $\alpha = 8$, $\beta = 11$ and $n = 15$, computed by means of the formulas in [50]. The exploration-exploitation tradeoff is clearly seen: Both for large p, where exploration dominates, and for small p, where exploitation dominates, the runtime behavior of the GHC is suboptimal. For a certain $p = p^*$, the expected first hitting time is minimized; in our example, this happens for $p^* \approx 0.215$, resulting in $E(T_1) \approx 234.3$. Note that we have set the parameter p to a fixed, time-independent value; gradually decreasing it would possibly produce improved performance.[8]

Unfortunately, it is rather the exception than the regular case that after a decomposition of the space of memory states into subsets, the Markov property remains valid for the subsets. The *level reaching* method for obtaining runtime bounds, which has been developed in articles on evolutionary algorithms (EAs) (cf. Droste et al. [12] or Borisovsky and Eremeev [6]), is applicable in a broader range of cases. The basic idea of this method is to define *subgoals* for the final goal of attaining a set $\mathcal{M}^*$ of memory states producing optimal solutions (or solutions considered as of sufficient quality). In our generic framework, the method can be formulated as follows: Let $\mathcal{M}$ be the set of all possible memory states. A hierarchy

$$\mathcal{H}_1 \supseteq \mathcal{H}_2 \supseteq \ldots \supseteq \mathcal{H}_K = \mathcal{M}^*$$

is defined, where $\mathcal{H}_k \subseteq \mathcal{M}$ stands for the set of all memory states on a certain "quality level" k, and reaching this quality level is considered as "subgoal" k ($k = 1, \ldots, K$). The algorithm under consideration must fulfill the *monotonicity* constraint $m_t \in \mathcal{H}_k \Rightarrow m_{t+1} \in \mathcal{H}_k$. The expected first hitting time

[8] For the runtime analysis of SA with decreasing temperature, cf. Wegener [62].

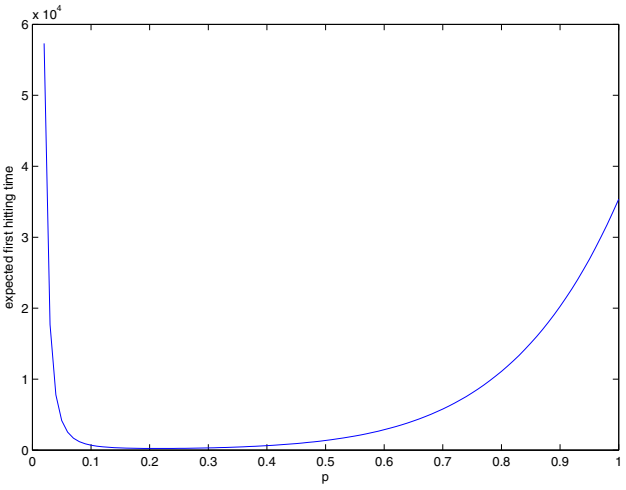


Fig. 6.1 Expected first hitting time of the GHC in dependence of the acceptance probability p for the illustration example.

t_k of subset $\mathcal{H}_k$, i.e., the expected time until subgoal k is reached for the first time, is given by $t_k = E(\min\{t : m_t \in \mathcal{H}_k\})$.

In the analysis of a considered algorithm on a special problem instance, one tries to identify upper bounds η_{ij} for the expected runtime until satisfying subgoal j, provided that the algorithm starts in an arbitrary memory state $m \in \mathcal{H}_i$. Then, e.g., it holds that

$$t_K \leq \sum_{i=1}^{K-1} \eta_{i,i+1}. \tag{6.10}$$

Often, the algorithm performs *independent* trials to achieve $\mathcal{H}_{i+1}$ from $\mathcal{H}_i$. Let in this situation $\alpha_{ij} > 0$ be a lower bound for the probability that $\mathcal{H}_j$ is reached after one iteration if the current state is in $\mathcal{H}_i$. Then, we can set $\eta_{i,i+1} = 1/\alpha_{i,i+1}$ and apply (6.10). Borisovsky and Eremeev [6] also provide stronger bounds on the values $(t_1, \ldots, t_K)$ derived from the matrix $A = (\alpha_{ij})$ in this more specific context.

A natural definition of subgoals derives from the possible cost function values: Let $\phi_1 > \ldots > \phi_K$ be the cost function values in decreasing order. Assume that the current memory state m_t contains x_t^{bsf} as a component. By defining $\mathcal{H}_k$ as the set of all memory states m for which $f(x^{bsf}) \leq \phi_k$, the required monotonicity property is satisfied. This principle has enabled the derivation of several runtime results for EAs (cf. [61, 51]).

Recently, it has been shown that the level-reaching approach can also be extended to *continuous* memory state spaces, as they occur in ACO, EDAs

or PSO. The article [31] provides general lemmas to ensure the mathematical validity of this extension, and derives expected first hitting time results for the MMAS variant of ACO on several standard test functions analyzed previously in the EA field.

(3) Martingales and Supermartingales

Let us consider again the Markov process (m_t) and assume that the currently proposed solution x_t is derived from the current state $m_t \in \mathcal{M}$, i.e., $x_t = \psi(m_t)$ with some function ψ. Furthermore, let us assume that a distance function d on $\mathcal{M}$ is given, which allows it in particular to define the distance $d(m, \mathcal{M}^*) = \min\{d(m, m^*) | m^* \in \mathcal{M}^*\}$ of a state m to the set $\mathcal{M}^*$ of memory states producing an optimal solution as the currently proposed solution. A possible way to define d may, e.g., consist in setting $d(m, m') = |f(\psi(m)) - f(\psi(m'))|$. For given $\mathcal{M}^*$, let us abbreviate $d(m, \mathcal{M}^*)$ by $d(m)$. We restrict the discussion to the case where $\mathcal{M}$ is a discrete finite set. He and Yao [36, 38] define the one-step *mean drift* in state m as the conditional expectation

$$E(d(m_t) - d(m_{t+1}) \,|\, m_t = m) = d(m) - \sum_{m'} p(m, m') d(m'),$$

where $p(m, m')$ is the transition probability from state m to state m'. In the case where the mean drift is always zero, the process $(d(m_t))$ is a *martingale*, which is a stochastic process (Y_t) with the property $E(Y_{t+1}|Y_1, \ldots, Y_t) = Y_t$. For problem instances, however, that are not *deceptive* in the sense of systematically misleading the search, it can happen that the drift is always positive. In this case, $(d(m_t))$ becomes a *supermartingale*, i.e., a process (Y_t) with $E(Y_{t+1}|Y_1, \ldots, Y_t) \leq Y_t$.

In [36, 38], drift analysis is applied to the runtime analysis of EAs on some special functions, and general conditions for classifying a problem instance as "easy" or "hard" for the considered algorithm are given, where "easy" and "hard" mean expected first hitting times of polynomial and exponential order in the problem instance size n, respectively. E.g., in [38], the following theorem is shown: A problem belongs to the "easy" class if and only if there exists a distance function $d(m)$ such that (i) $d(m_t) \leq g_1(n)$ with a polynomial $g_1(n)$, and $E(d(m_t) - d(m_{t+1}) \,|\, m_t) \geq c_{low}$ with a constant $c_{low} > 0$ for any state m_t in any iteration t.

(4) ODE Approximations

The scenario of a continuous memory state space $\mathcal{M}$ causes particular problems for the analysis, especially in the case where the search mechanism does not rely on the best-so-far solution x_t^{bsf}, but on other parts of the current state, such that in a natural definition of the sets $\mathcal{H}_k$ defining subgoals, the monotonicity property mentioned above is not satisfied anymore. An example is the Ant System variant of ACO, introduced in [10]. In the analysis of this algorithm, it is not clear how to define subgoals in a helpful way.

A tool to enable a mathematical runtime analysis also in such cases is the asymptotic approximation of the stochastic process by a limiting process obtained by letting some parameter tend to a boundary value. In the Ant System case, this can be done for the evaporation rate ρ, as shown in [25, 27]: If $\rho \to 0$ (which is a meaningful asymptotic for ACO in view of one of the convergence results outlined in Section 6.4 which has $\rho_t \to 0$), the Ant System process approaches a limiting process where the pheromone vector (i.e., the current memory state) follows a *deterministic* dynamic, described by a system of *ordinary differential equations* (ODEs). On the other hand, the sample points in L_t still remain stochastic, which means that the explorative capacity of the algorithm is not reduced. Conclusions on convergence and on expected first hitting time for special test functions can be derived, see [25, 27]. A similar approach has been presented in [52]. This technique is still relatively new in the literature on analysis of metaheuristics, such that its potential will yet have to be explored in the future.

6.7 Conclusions

In this paper, several notions of convergence in the context of optimization by metaheuristics have been discussed, and some fundamental mathematical proof ideas for showing convergence of special metaheuristic algorithms for all CO problems have been presented. In particular, we have distinguished between a weaker form of convergence termed best-so-far convergence, and a stronger form termed model convergence. Examples of algorithms owing one or both of these types of convergence properties have been given.

Let us shortly outline some open research topics. Several metaheuristic variants have not even been shown to converge in a best-so-far sense; as an example, let us mention ACO variants without pheromone bounds and working only with iteration-best reinforcement. It would be interesting to find out under which conditions these variants converge to the optimum at all. An obvious other topic of future research is of course to strengthen existing convergence results of best-so-far type, as available for a large class of other metaheuristic variants, to results of model convergence type.

Another open problem has already been outlined at the end of Subsection 6.4.2: the investigation of the connections between convergent or "subconvergent" parameter schemes for an algorithm to its performance within finite time (measured, e.g., by the average achieved solution quality).

In the field of convergence speed analysis, it seems that at the moment, the area of open problems is much larger that that of available results. Besides the challenging goal of analyzing the performance of metaheuristics on NP-complete problems, it may also be important to strive for more *general* results than those available at present. Although no-free-lunch theorems (cf. Section 6.6) set limits to generalizability, one might attempt to identify

runtime-relevant properties common to larger classes of different problems (e.g., fitness landscape properties) and to study the impacts of these properties on the runtime behavior in depth.

A final remark concerns "mat-heuristic" algorithms combining a metaheuristic approach with mathematical programming techniques. At the moment, convergence results for such algorithms (as an example, let us mention the well-known Local Branching algorithm by Fischetti and Lodi [16]) seem to be yet unavailable. As mentioned in the introduction, a rigorous mathematical analysis of algorithms would be especially desirable in this field overlapping with that of traditional mathematical optimization.

References

1. E.H.M. Aarts and J.H.L. Korst. *Simulated Annealing and Boltzmann Machines*. John Wiley and Sons, Chichester, UK, 1990.
2. M.H. Alrefaei and S. Andradóttir. A simulated annealing algorithm with constant temperature for discrete stochastic optimization. *Management Science*, 45:748–764, 1999.
3. L. Bianchi, M. Dorigo, L.M. Gambardella, and W.J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, to appear.
4. M. Birattari, P. Balaprakash, and M. Dorigo. The ACO/F-Race algorithm for combinatorial optimization under uncertainty. In K.F. Doerner, M. Gendreau, P. Greistorfer, W.J. Gutjahr, R.F. Hartl, and M. Reimann, editors, *Metaheuristics—Progress in Complex Systems Optimization*, pages 189–203. Springer Verlag, Berlin, Germany, 2006.
5. Y. Borenstein and R. Poli. Information perspective of optimization. In T.P. Runarsson, H.-G. Beyer, E.K. Burke, J.J. Merelo Guervós, L.D. Whitley, and X. Yao, editors, *Proceedings of the 9th Conference on Parallel Problem Solving from Nature*, volume 4193 of *Lecture Note in Computer Science*, pages 102–111. Springer Verlag, Berlin, Germany, 2006.
6. P.A. Borisovsky and A.V. Eremeev. A study on the performance of the (1+1)-evolutionary algorithm. In K.A. De Jong, R. Poli, and J.E. Rowe, editors, *Proceedings of Foundations of Genetic Algorithms*, volume 7, pages 271–287. Morgan Kaufmann Publishers, San Mateo, CA, 2003.
7. J. Brimberg, P. Hansen, and N. Mladenović. Convergence of variable neighborhood search. Les Cahiers du GERAD G-2002-21, Groupe d'études et de recherche en analyse des décisions (GERAD), Montréal, Canada, 2002.
8. T. Homem de Mello. Variable-sample methods for stochastic optimization. *ACM Transactions on Modeling and Computer Simulation*, 13:108–133, 2003.
9. F. Van den Bergh and A.P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176:937–971, 2006.
10. M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 26:1–13, 1996.
11. S. Droste, T. Jansen, and I. Wegener. Perhaps not a free lunch but at least a free appetizer. In W. Banzhaf, J.M. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M.J. Jakiela, and R.E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference 1999*, pages 833–839. Morgan Kaufmann, San Mateo, CA, 1999.
12. S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

13. S. Droste, T. Jansen, and I. Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39(4):525–544, 2006.
14. T. English. Optimization is easy and learning is hard in the typical function. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 2, pages 924–931. IEEE Press, Piscataway, NJ, 2000.
15. T. English. On the structure of sequential search: beyond no free lunch. In J. Gottlieb and G.R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization, 4th European Conference, EvoCOP 2004*, volume 3004 of *Lecture Notes in Computer Science*, pages 95–103. Springer Verlag, Berlin, Germany, 2004.
16. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming Ser. B*, 98:23–47, 2003.
17. S.B. Gelfand and S.K. Mitter. Analysis of simulated annealing for optimization. In *Proceedings of the 24th IEEE Conference on Decision and Control*, pages 779–786, 1985.
18. S.B. Gelfand and S.K. Mitter. Simulated annealing with noisy or imprecise measurements. *Journal of Optimization Theory and Applications*, 69:49–62, 1989.
19. F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Springer, 2003.
20. C. Gonzalez, J.A. Lozano, and P. Larrañaga. Analyzing the PBIL algorithm by means of discrete dynamical systems. *Complex Systems*, 11:1–15, 1997.
21. W.J. Gutjahr. A graph–based ant system and its convergence. *Future Generation Computer Systems*, 16:873–888, 2000.
22. W.J. Gutjahr. ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters*, 82:145–153, 2002.
23. W.J. Gutjahr. A converging ACO algorithm for stochastic combinatorial optimization. In A.A. Albrecht and K. Steinhöfel, editors, *Stochastic Algorithms: Foundations and Applications, Second International Symposium, SAGA 2003*, volume 2827 of *Lecture Notes in Computer Science*, pages 10–25. Springer Verlag, Berlin, Germany, 2003.
24. W.J. Gutjahr. An ant-based approach to combinatorial optimization under uncertainty. In M. Dorigo, L. Gambardella, F. Mondada, T. Stützle, M. Birratari, and C. Blum, editors, *ANTS'2004, Fourth International Workshop on Ant Algorithms and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 238–249. Springer Verlag, Berlin, Germany, 2004.
25. W.J. Gutjahr. On the finite-time dynamics of ant colony optimization. *Methodology and Computing in Applied Probability*, 8:105–133, 2006.
26. W.J. Gutjahr. Mathematical runtime analysis of ACO algorithms: survey on an emerging issue. *Swarm Intelligence*, 1:59–79, 2007.
27. W.J. Gutjahr. First steps to the runtime complexity analysis of ant colony optimization. *Computers & Operations Research*, 35:2711–2727, 2008.
28. W.J. Gutjahr. Stochastic search in metaheuristics. Technical report, Department of Statistics and Decision Support Systems, University of Vienna, 2008.
29. W.J. Gutjahr, S. Katzensteiner, and P. Reiter. A VNS algorithm for noisy problems and its application to project portfolio analysis. In J. Hromkovic, R. Královic, M. Nunkesser, and P. Widmayer, editors, *Stochastic Algorithms: Foundations and Applications, Second International Symposium, SAGA 2007*, volume 4665 of *Lecture Notes in Computer Science*, pages 93–104. Springer Verlag, Berlin, Germany, 2007.
30. W.J. Gutjahr and G. Pflug. Simulated annealing for noisy cost functions. *Journal of Global Optimization*, 8:1–13, 1996.
31. W.J. Gutjahr and G. Sebastiani. Runtime analysis of ant colony optimization with best-so-far reinforcement. *Methodology and Computing in Applied Probability*, 10:409–433, 2008.
32. B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13:311–329, 1988.

33. P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.
34. R.F. Hartl. A global convergence proof for a class of genetic algorithms. Technical report, Institut für Ökonometrie & Operations Research, Technische Universität Wien, 1990.
35. J. He and X. Yao. Conditions for the convergence of evolutionary algorithms. *Journal of Systems Architecture*, 47:601–612, 2001.
36. J. He and X. Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127:57–85, 2003.
37. J. He and X. Yao. Towards an analytic framework for analysing the computation time of evolutionary algorithms. *Artificial Intelligence*, 145:59–97, 2003.
38. J. He and X. Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3:21–35, 2004.
39. H.H. Hoos. On the runtime behavior of stochastic local search algorithms for SAT. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 661–666. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1999.
40. H.H. Hoos and T. Stützle. Local search algorithms for SAT: an empirical investigation. *Journal of Automated Reasoning*, 24:421–481, 2000.
41. C. Igel and M. Toussaint. On classes of functions for which no free lunch results hold. *Information Processing Letters*, 86:317–321, 2003.
42. C. Igel and M. Toussaint. A no-free-lunch theorem for non-uniform distributions of target functions. *Journal of Mathematical Modelling and Algorithms*, 3:313–322, 2004.
43. S.H. Jacobson, K.A. Sullivan, and A.W. Johnson. Discrete manufacturing process design optimization using computer simulation and generalized hill climbing algorithms. *Engineering Optimization*, 31:147–260, 1998.
44. S.H. Jacobson and E. Yuecesan. Analyzing the performance of generalized hill climbers. *Journal of Heuristics*, 10:387–405, 2004.
45. J. Jaegerskuepper. Lower bonds for hit-and-run direct search. In J. Hromkovic, R. Královic, M. Nunkesser, and P. Widmayer, editors, *Stochastic Algorithms: Foundations and Applications, Second International Symposium, SAGA 2007*, volume 4665 of *Lecture Notes in Computer Science*, pages 118–129. Springer Verlag, Berlin, Germany, 2007.
46. Y. Jin and J. Branke. Evolutionary optimization in uncertain environments—a survey. *IEEE Transactions on Evolutionary Computation*, 9:303–317, 2005.
47. J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948. IEEE Press, Piscataway, NJ, 1995.
48. J. Kennedy and R.C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, volume 5, pages 4104–4109. IEEE Press, Piscataway, NJ, 1997.
49. L. Margolin. On the convergence of the cross-entropy method. *Annals of Operations Research*, 134:201–214, 2005.
50. H. Muehlenbein and J. Zimmermann. Size of neighborhood more important than temperature for stochastic local search. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 2, pages 1017–1024. IEEE Press, Piscataway, NJ, 2000.
51. P.S. Oliveto, J. He, and X. Yao. Time complexity of evolutionary algorithms for combinatorial optimization: a decade of results. *International Journal of Automation and Computing*, 4:281–293, 2007.
52. P. Purkayastha and J.S. Baras. Convergence results for ant routing algorithms via stochastic approximation and optimization. In *Proceedings of the 46th IEEE Conference on Decision and Control*, pages 340–345. IEEE Press, Piscataway, NJ, 2007.
53. R.Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, pages 127–170, 1999.
54. G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5:96–101, 1994.

55. G. Sebastiani and G.L. Torrisi. An extended ant colony algorithm and its convergence analysis. *Methodology and Computing in Applied Probability*, 7:249–263, 2005.
56. J.C. Spall, S.D. Hill, and D.R. Stark. Theoretical framework for comparing several stochastic optimization algorithms. In G. Calafiore and F. Dabbene, editors, *Probabilistic and Randomized Methods for Design under Uncertainty*, pages 99–117. Springer Verlag, London, UK, 2006.
57. T. Stützle and M. Dorigo. A short convergence proof for a class of ACO algorithms. *IEEE Transactions on Evolutionary Computation*, 6:358–365, 2002.
58. T. Stützle and H.H. Hoos. $\mathcal{MAX} - \mathcal{MIN}$ ant system. *Future Generation Computer Systems*, 16:889–914, 2000.
59. A.S. Thikomirov. On the convergence rate of the Markov homogeneous monotone optimization method. *Computational Mathematics and Mathematical Physics*, 47:817–828, 2007.
60. I.C. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85:317–325, 2003.
61. I. Wegener. Methods for the analysis of evolutionary algorithms on pseudo-boolean functions. In R. Sarker, M. Mohammadia, and X. Yao, editors, *Evolutionary Optimization*, volume 48 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Norwell, MA, 2003.
62. I. Wegener. Simulated annealing beats metropolis in combinatorial optimization. In L. Caires, G.F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 589–601. Springer Verlag, Berlin, Germany, 2005.
63. D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.
64. Y. Yu and Z.-H. Zhou. A new approach to estimating the expected first hitting time of evolutionary algorithms. In *Proceedings of the Twentyfirst National Conference on Artificial Intelligence*, pages 555–560. AAAI Press / The MIT Press, Menlo Park, CA, USA, 2006.
65. M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization: a critical survey. *Annals of Operations Research*, 131:373–379, 2004.

Chapter 7
MIP-based GRASP and Genetic Algorithm for Balancing Transfer Lines

Alexandre Dolgui, Anton Eremeev, and Olga Guschinskaya

Abstract In this chapter, we consider a problem of balancing transfer lines with multi-spindle machines. The problem has a number of distinct features in comparison with the well-studied assembly line balancing problem, such as parameterized operation times, non-strict precedence constraints, and parallel operations execution. We propose a mixed-integer programming (MIP)-based greedy randomized adaptive search procedure (GRASP) and a genetic algorithm (GA) for this problem using a MIP formulation. Both algorithms are implemented in GAMS using the CPLEX MIP solver and compared to problem-specific heuristics on randomly generated instances of different types. The results of computational experiments indicate that on large-scale problem instances the proposed methods have an advantage over the methods from literature for finding high quality solutions. The MIP-based recombination operator that arranges the elements of parent solutions in the best possible way is shown to be useful in the GA.

7.1 Introduction

The problem considered in this chapter consists in balancing a transfer line where multi-spindle transfer machines without intermediate buffers are used. This problem is referred to as the *transfer line balancing problem* (TLBP) [5]. Machining transfer lines are usually paced and serial. They consist of a sequence of stations linked by an automated material handling device. In

Alexandre Dolgui · Olga Guschinskaya
Ecole Nationale Supérieure des Mines de Saint Etienne, Saint Etienne, France
e-mail: {dolgui,guschinskaya}@emse.fr

Anton Eremeev
Omsk Branch of Sobolev Institute of Mathematics SB RAS, Omsk, Russia
e-mail: eremeev@ofim.oscsbras.ru

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_7,

lines of this type, each station is equipped by a special machine-tool which performs machining operations block by block. All operations of each block are executed simultaneously using one multi-spindle head. The parallel execution of operations in a block is possible due to the fact that the multi-spindle heads carry several simultaneously activated tools. When machining at the current station is finished (all blocks installed on this machine have been activated) the part is moved to the next station. The time span between two movements can not exceed the given time value T_0 referred to as *line cycle time*. The balancing problem consists in assigning the given set of operations to parallel blocks and stations under given assignment restrictions.

The line balancing problem in the assembly environment is well-studied in the literature. Several reviews of different formulations and used solution methods are available e.g. in [1, 2, 9]. The TLBP has a number of unique characteristics such as parameterized operation times, non-strict precedence constraints, and parallel operations execution. These features make it impossible to use directly the optimization methods developed for assembly line balancing problems; for details see [5]. Several exact (e.g. mixed-integer programming and graph approaches) and heuristic (e.g. FSIC (First Satisfy Inclusion Constraints) and multi-start decomposition algorithm) methods have been developed for the TLBP. A description of these methods is given in [10]. Later, in [11] it was proposed to use greedy randomized adaptive search procedures (GRASP) for solving this problem.

In this chapter, we propose a MIP-based greedy randomized adaptive search procedure (GRASP) and a genetic algorithm (GA) for the TLBP, using the MIP formulation [5] of the TLBP in both algorithms. The solution construction and the local improvement stages of GRASP are based on solving sub-problems of smaller size. The same solution construction method is used for building the initial population in the GA. The crossover and mutation in the GA are combined in a MIP-recombination operator, similar to the recombination proposed in [3].

Both algorithms are implemented in GAMS using the CPLEX MIP solver and compared to problem-specific heuristics [10] on randomly generated instances of different types. The results of computational experiments indicate that on large problem instances the methods proposed offer an advantage over the methods from literature in finding high quality solutions. The capability of the MIP-recombination operator to arrange the elements of parent solutions in the best possible way is shown to be useful in the GA.

The chapter is organized as follows. The problem statement in the MIP formulation is given in Section 7.2. The solution methods are discussed in Sections 7.3 and 7.4. The results of computational experiments are presented in Section 7.5. Concluding remarks are given in Section 7.6.

7.2 Problem Statement

For the TLBP the following input data are assumed to be given [5]:

- $\mathbf{N}$ is the set of all operations involved in machining of a part;
- T_0 is the maximal admissible line cycle time;
- τ^S and τ^b are the auxiliary times needed for activation of a station and a spindle head (block), respectively;
- C_1 and C_2 are the relative costs of one station and one spindle head (block);
- m_0 is the maximal admissible number of stations;
- n_0 is the maximal number of spindle heads (blocks) per station;
- *Precedence constraints* between the operations. These constraints define a non-strict partial order relation over the set of operations $\mathbf{N}$. They are represented by a digraph $G = (\mathbf{N}, D)$. An arc $(i, j) \in \mathbf{N}^2$ belongs to the set D if and only if the block with operation j cannot precede the block with operation i. (If $(i, j) \in D$ then the operations i and j can be performed simultaneously in a common block.)
- *Inclusion constraints* defining the groups of operations that must be assigned to the same station, because of a required machining tolerance. These constraints can be represented by a family ES of subsets of $\mathbf{N}$, such that all operations of the same subset $e \in ES$ must be assigned to the same station;
- *Station exclusion constraints* defining the groups of operations that cannot be assigned to the same station because of their technological incompatibility. These constraints are represented by a family $\overline{ES}$ of subsets of $\mathbf{N}$, such that all elements of the same subset $e \in \overline{ES}$ cannot be assigned to the same station.
- *Block exclusion constraints* defining the groups of operations that cannot be assigned to the same block because of their technological incompatibility. These constraints are represented by a family $\overline{EB}$ of subsets from $\mathbf{N}$, such that all elements of the same subset $e \in \overline{EB}$ cannot be assigned to the same block.
- For each operation j, its processing time t_j is given or, alternatively, it may be characterized by two parameters: the required working stroke length λ_j and the maximal admissible feed per minute s_j. The working stroke length includes the required depth of cut and the distance between the tool and the part surface.

A MIP formulation for solving the TLBP was suggested in [5]. Here, we reproduce this model with the improvements proposed in [10].

Let us denote the set of all operations assigned to station k by N_k and let the set of operations grouped into block l of station k be N_{kl}. Station processing time $t^S(N_k)$ equals the sum of its block processing times: $t^S(N_k) = \sum_{l=1}^{n_k} t^b(N_{kl}) + \tau^S$.

We will consider two definitions of block processing time. A simplified definition [10] uses the assumption that the block processing time $t^b(N_{kl})$ is

equal to the duration of the longest operation in the block:

$$t^b(N_{kl}) = \max\{t_j | j \in N_{kl}\} + \tau^b. \tag{7.1}$$

A more general definition [12] does not use the processing times of operations t_j, but rather the parameters λ_j and s_j:

$$t^b(N_{kl}) = \frac{\max\{\lambda_i | i \in N_{kl}\}}{\min\{s_i | i \in N_{kl}\}} + \tau^b. \tag{7.2}$$

Note that the latter definition covers the first case, assuming $\lambda_j = t_j,\ s_j = 1$ for all j. Besides that, it suits better the practical situations where the operations assigned to several tools fixed within one block may require different depths of their cuts and different maximal admissible feed speeds.

In the MIP formulation below we use the following notation:

- q is the block index; $q = (k-1)n_0 + l$ is the l-th block of a station k;
- q_0 is the maximal possible value of q, $q_0 = m_0 n_0$;
- $S(k) = \{(k-1)n_0 + 1, \ldots, kn_0\}$ is the set of block indices for a station k;
- $Q(j)$ is the set of indices q (blocks) where operation j can be assigned;
- $K(j)$ is the set of indices k (stations) where operation j can be assigned;
- e is a set of operations which is an element of ES, $\overline{ES}$ or $\overline{EB}$;
- $j(e)$ is an arbitrarily fixed operation from the set e.
- t_j is the execution time of operation j if it is performed alone in a block. In the simplified formulation, t_j is given as input data. Otherwise, $t_j = \frac{\lambda_j}{s_j}$.
- $t_{ij} = \frac{\max\{\lambda_i, \lambda_j\}}{\min\{s_i, s_j\}}$ is the execution time of two operations i, j if they are performed in one block. In the simplified formulation this value is not used.

The following variables will be involved:

- X_{jq} is a binary decision variable (1 if operation j is assigned to block q and 0 otherwise);
- F_q is an auxiliary real-valued variable for determining the time of processing the block q;
- Y_q is an auxiliary binary variable that indicates if the block q exists;
- Z_k is an auxiliary binary variable that indicates if the station k exists.

The variables Y_q and Z_k are used to count the number of blocks and stations, respectively. To reduce the number of decision variables and constraints, the sets of possible block and station indices $Q(j)$, $K(j)$ for each operation j are obtained by means of the procedure described in [5].

The problem consists in the minimization of investment costs incurred by construction of stations and spindle heads (blocks):

$$\text{Min } C_1 \sum_{k=1}^{m_0} Z_k + C_2 \sum_{q=1}^{q_0} Y_q \tag{7.3}$$

subject to

$$\sum_{q \in Q(j)} (q-1) X_{jq} \geq \sum_{s \in Q(i)} (s-1) X_{is}, \quad (i,j) \in D, \tag{7.4}$$

$$\sum_{q \in Q(j)} X_{jq} = 1, \quad j \in \mathbf{N}, \tag{7.5}$$

$$\sum_{j \in e \setminus \{j(e)\}} \sum_{q \in S(k) \cap Q(j)} X_{jq} = (|e|-1) \sum_{q \in S(k)} X_{j(e)q}, \quad e \in ES,\ k \in K(j(e)), \tag{7.6}$$

$$\sum_{j \in e} X_{jq} \leq |e| - 1, \quad e \in \overline{EB},\ q \in \cap_{j \in e} Q(j), \tag{7.7}$$

$$\sum_{j \in e} \sum_{q \in S(k) \cap Q(j)} X_{jq} \leq |e| - 1, \quad e \in \overline{ES},\ k \in \cap_{j \in e} K(j), \tag{7.8}$$

$$F_q \geq (t_i + \tau^b) X_{iq}, \quad i \in \mathbf{N},\ q \in Q(i), \tag{7.9}$$

$$F_q \geq (t_{ij} + \tau^b)(X_{iq} + X_{jq} - 1), \quad i,j \in \mathbf{N},\ i < j,\ q \in Q(i) \cap Q(j), \tag{7.10}$$

$$\tau^S + \sum_{q \in S(k)} F_q \leq T_0, \quad k = 1, 2, \ldots, m_0, \tag{7.11}$$

$$Y_q \geq X_{jq}, \quad j \in \mathbf{N},\ q \in Q(j), \tag{7.12}$$

$$Z_k = Y_{(k-1)n_0+1}, \quad k = 1, 2, \ldots, m_0, \tag{7.13}$$

$$Y_{q-1} - Y_q \geq 0, \quad q \in S(k) \setminus \{(k-1)n_0 + 1\},\ k = 1, 2, \ldots, m_0, \tag{7.14}$$

$$Z_{k-1} - Z_k \geq 0, \quad k = 2, 3, \ldots, m_0, \tag{7.15}$$

$$X_{jq},\ Y_q,\ Z_k, \in \{0,1\}, \quad j \in \mathbf{N},\ q = 1, 2, \ldots, q_0,\ k = 1, \ldots, m_0, \tag{7.16}$$

$$F_q \in [0, T_0 - \tau^S - \tau^b], \quad q = 1, 2, \ldots, q_0. \tag{7.17}$$

Here inequalities (7.4) impose the precedence constraints; equalities (7.5) reflect the fact that each operation must be assigned to exactly one block; constraints (7.6) determine the necessity of grouping certain operations in the same station; constraints (7.7)-(7.8) deal with the impossibility of grouping certain operations in one block or executing certain operations at the same station, respectively; constraints (7.9) and (7.10) determine the block processing times according to (7.1) or (7.2): here condition (7.9) corresponds to the case of a single operation in a block, while (7.10) covers the cases of two or more operations (note that in the simplified formulation with block time defined by (7.1), inequality (7.10) is redundant); constraint (7.11) imposes the bound on cycle time; constraints (7.12) ensure that block q exists in the design decision if and only if $X_{jq} = 1$ for some j; equalities (7.13) ensure that a station k exists in the design decision if and only if at least one block is assigned to it; constraints (7.14) guarantee that block q is created in station k only if block $q-1$ exists for this station; constraints (7.15) ensure that station k can be created only if station $k-1$ is created.

Inequalities (7.14) and (7.15) mainly serve as symmetry-breaking cuts in this model (note that by a simple modification of (7.13) one could make these inequalities redundant). Bounds (7.16) are also imposed to reduce the polyhedron of the linear relaxation. One could assume that all variables Y_q and Z_k are real values from the interval [0,1] but we do not use this assumption, because our preliminary experiments indicate that binary variables Y_q and Z_k yield a more appropriate problem formulation for the CPLEX MIP solver.

Another modification that can improve the performance of branch-and-cut algorithms consists in adding a relatively small penalty term to the objective function. The greater is the block number where an operation is assigned, the greater penalty is given:

$$\text{Min } C_1 \sum_{k=1}^{m_0} Z_k + C_2 \sum_{q=1}^{q_0} Y_q + C_3 \sum_{q=1}^{q_0} \sum_{j \in \mathbf{N}} (1-q) X_{jq}. \tag{7.18}$$

Here, the weight C_3 may be chosen sufficiently small, so that the optimal solution of the modified problem (7.4)–(7.18) is also optimal for problem (7.3)–(7.17). The penalty term breaks some symmetries of the problem and provides appropriate bias when the branching is made. Our experiments indicate that for faster search of approximate solutions the value C_3 may be chosen adaptively (see the details in Section 7.5).

7.3 Greedy Randomized Adaptive Search Procedure

GRASP is a multi-start metaheuristic algorithm, where each iteration consists of two phases: constructing a feasible solution and improving it. Both phases are repeated interchangeably until a stopping criterion is satisfied. Extensive bibliographies on GRASP were published in [8, 16]. The general scheme of GRASP is as follows:

GRASP method
Until a stopping criterion is satisfied do:
1. Construct a random feasible solution.
2. If a feasible solution is constructed,
   apply a local improvement procedure to it.
3. Update the best found solution.

7.3.1 Construction Phase

In the case of the TLBP, a feasible solution at Step 1 can be obtained by applying a randomized greedy heuristic algorithm (see, e.g., [13]). The MIP-based greedy algorithm for the TLBP starts from a transfer line with an empty set of blocks, and then the blocks are created consecutively. A feasible solution is constructed by adding a set N^t_{add} of one or more operations at each step t of the greedy algorithm. We will denote by N^t the set of operations that have been assigned to stations on steps $1, \ldots, t$ assuming $N^0 = \emptyset$.

The set N^t_{add} is constructed by a randomized procedure which has two tunable parameters $\alpha \in [0, 1]$ and $\beta \in \{1, \ldots, |\mathbf{N}|\}$. Adjustment of α and β will be discussed in Section 7.5. At the beginning of step t it is assumed that $N^t_{add} = \emptyset$, then it is extended in the following loop:

1. Compute the *set of candidate operations* N_{CL}, consisting of all operations that can be allocated after the set of operations $N^{t-1} \cup N^t_{add}$ in view of inclusion and precedence constraints. To this end, we use a supplementary digraph $G' = (\mathbf{N}, D')$ which is obtained from G by adding the arcs (i, i') for all pairs i, i' such that $i \in e,\ i' \in e$ for some $e \in ES$, and by taking the transitive closure of this extended digraph. The inclusion and precedence constraints will not be violated if a new operation j is executed after all operations of the set $N^{t-1} \cup N^t_{add}$, provided that $N^{t-1} \cup N^t_{add}$ contains all such i that $(i, j) \in D'$ and $(j, i) \notin D'$. The graph G' may be computed by means of Warshall's algorithm in time $O(|\mathbf{N}|^3)$, before the GRASP iterations begin.
2. Rank the operations in N_{CL} according to the values of *greedy function* $g(j)$ (this function will be described later). Find

$$g_{max} = \max\{g(j) : j \in N_{CL}\} \text{ and } g_{min} = \min\{g(j) : j \in N_{CL}\}.$$

3. Place the well-ranked candidate operations j with $g(j) \geq g_{max} - \alpha(g_{max} - g_{min})$ into a set N_{RCL}, called *restricted candidate list*. The parameter α controls the trade-off between randomness and greediness in the construction process.
4. Select an element j uniformly at random from N_{RCL} and add j to the set N^t_{add}.
5. Include into N^t_{add} all operations i such that $(i, j) \in D'$ and $(j, i) \in D'$ (the precedence and inclusion constraints imply that these operations must be placed in the same block with j).

This loop continues until β iterations are made or there are no more operations to add, i.e. $N^{t-1} \cup N^t_{add} = \mathbf{N}$.

The iterations of the greedy algorithm continue until either all operations are assigned and a feasible solution is obtained or it is impossible to create a new station since $m + 1 > m_0$.

The greedy function $g(j)$ measures the impact of assigning operation j at the current iteration. Several greedy heuristics were elaborated for the simple assembly line balancing problem with the greedy functions based on priority rules; see, e.g., [17]. However, all previously considered priority rules are based on the hypothesis that all operations are executed sequentially and the operation times are cumulated. This hypothesis is not right for the TLBP, where operations can be executed in parallel. As a consequence, the greedy function can hardly be based on the known priority rules. We use a simple greedy function $g(j)$ equal to the lower bound on the number of blocks required to assign all successors of operation j. This lower bound is calculated by the algorithm suggested in [5].

Once the set N^t_{add} is chosen, the operations of this set are appended to the current partial solution which has been computed on the previous iterations. Let us define for all $j \in \mathbf{N}$, $q = 1, \ldots, q_0$ the set of values $x^{(t)}_{jq}$, such that $x^{(t)}_{jq} = 1$ if operation j is assigned to block q in the partial solution obtained at iteration t, and $x^{(t)}_{jq} = 0$ otherwise. Allocation of the new operations can be carried out by means of a supplementary MIP problem. This MIP problem is formulated by the set of constraints (7.4)–(7.17) and the objective function (7.18) but a large number of binary variables are fixed equal to zero as described below.

Let k_{use} denote the number of the last station to which operations have been assigned at the latest partial solution, i.e.

$$k_{use} = \max\left\{ k \mid \sum_{j \in \mathbf{N}} \sum_{q \in S(k)} x^{(t-1)}_{jq} \geq 1 \right\}.$$

We aim to allocate the operations of the set N^t_{add} to the stations with numbers not greater than $k_{max} = k_{use} + \beta$ (this is always possible if the problem is solvable and $k_{max} \leq m_0$). To this end, we do not fix the variables X_{jq} with $j \in N^t_{add}$ and $q \leq q_{max}$, where $q_{max} = k_{max} n_0$. We also do not fix the variables X_{jq} such that $x^{(t-1)}_{jq} = 1$ or $q = 1 + \max\{q | \sum_{j \in \mathbf{N}} x^{(t-1)}_{jq} \geq 1\}$ to allow some previously allocated operations to be moved into the first new block, if it allows to save the cost. All the rest of the variables X_{jq} are fixed to zero value. The resulting sub-problem at each step t of the greedy heuristic is solved by a MIP solver. The value of parameter β is chosen experimentally so that the resulting sub-problems involve as many operations as possible, but the computational cost of the MIP solver in each iteration t is "not too large".

7.3.2 Improvement Phase

The improvement heuristic starts with a feasible solution obtained at the construction phase in order to improve it. For this purpose, a MIP-based modification of the *decomposition algorithm with aggregate solving of sub-problems* (DAASS) [12] is used. The algorithm DAASS has already been used with heuristic FSIC [4], which constructs a feasible solution without applying any greedy function, in [10, 12].

The decomposition consists in cutting the sequence of stations corresponding to the given feasible solution into several non-intersecting subsequences. The size of each subsequence is chosen at random, as it is described below. The total number w of such subsequences is known only at the end of the decomposition procedure. A subsequence $K_r,\ r = 1, \ldots, w$ involving a random number of stations k_r, is used to generate a sub-problem SP_r.

In DAASS, each sub-problem SP_r is solved exactly by the graph approach described in [6]. The results of previously solved sub-problems in DAASS are taken into account while solving the next sub-problem: each block of operations existing in the solution to SP_r is replaced by a *macro-operation* and all macro-operations are included in the consecutive sub-problem SP_{r+1}.

In contrast to the DAASS method, the local improvement procedure used in the present chapter is based on solving a series of sub-problems in MIP formulation (7.4)–(7.18). To simplify the algorithm, at step $r,\ r = 1, \ldots, w$ we do not construct the macro-operations in this heuristic, but simply fix all binary variables non-related to the stations of set K_r, equal to the corresponding values of the best found solution. The remaining variables $X_{jq},\ Y_q,\ q \in \cup_{k \in K_r} S(k)$, and $Z_k, k \in K_r$ are optimized by a MIP solver.

In the randomized choice of the sets K_r we have to ensure that on one hand, the size of a sub-problem is not "too small" and the solver can often improve the heuristic solution, on the other hand, the size of a sub-problem is not "too large" and it is possible to apply the solver in reasonable CPU time.

The following parameters, are used to limit the size of the sub-problems [12]: (i) the maximal number of stations k_{max} within one subsequence, that is the maximum possible value of k_r; (ii) the maximal number of operations n_{max} within one subsequence.

The value k_r is chosen uniformly at random within $[1, k_{max}]$ and then can be modified so that the total number of operations in the sub-problem does not exceed N_{max} and $\sum_{r=1}^{w} k_r$ is not greater than the number of stations in the current heuristic solution.

7.4 Genetic Algorithm

A *genetic algorithm* is a random search method that models a process of evolving a population of individuals [14, 15]. Each individual corresponds to some solution of the problem (feasible or infeasible) and it is characterized by the *fitness* which reflects the objective function value and the satisfaction of problem constraints. The better the fitness value, the more chances are given for the individual to be selected as a parent. New individuals are built by means of a *reproduction* operator that usually consists of *crossover* and *mutation* procedures. The crossover procedure produces the offspring from two parent individuals by combining and exchanging their elements. The mutation procedure adds small random changes to an individual. The formal scheme of the GA with steady state replacement is as follows:

Steady-state scheme of the GA
1. Generate the initial population.
2. Assign $t := 1$.
3. Until a termination condition becomes true do:
   3.1 Selection: choose p_1, p_2 from the population.
   3.2 Produce a child c applying mutation and crossover to p_1 and p_2. The crossover is used with probability P_c.
   3.3 Choose the worst individual in population w.r.t. the fitness function and replace it by c.
   3.4 t:=t+1.
4. Result is the best found solution w.r.t. fitness function.

In our implementation of the GA the fitness function is identical with the objective function. The choice of each parent on Step 3.1 is done by the s-tournament selection: take s individuals at random from the population (uniformly distributed, repetitions allowed) and select the best one w.r.t. the objective function. The population size remains constant during the execution of the GA - this parameter is denoted by N_{ind}.

In each iteration of a steady-state GA, most of the individuals of the current population are kept unchanged, which is different from the canonical GA proposed by Holland [14]. In many implementations of the steady-state GA the offspring replace the worst individuals of the population, but there are alternative replacement rules as well [15].

In this chapter, we will assume that the GA is restarted every time the termination condition halts it. This continues until the overall execution time will reach the limit T. The best solution found over all runs is returned as the final output. In our computational experiments we have tested an alternative approach, where the GA runs for the whole period T without restarts (see Subsection 7.5.2) but it turned to be inferior to the GA with this restart rule.

Let θ be the iteration when the latest solution improvement took place. The termination condition of GA is: restart if during the last θ iterations there was no best-found solution improvement and $t > N_{ind}$.

Encodings of solutions in a GA are usually called *genotypes*. One of the most essential issues in the development of a GA is the choice of representation of solutions in genotypes. In the GA proposed in this chapter, the genotype consists of values of the binary variables X_{jq} which describe the whole assignment of operations. The genotypes of the initial population are generated at random by N_{ind} runs of the GRASP heuristic described in Section 7.3.

MIP-Recombination

As proposed in [3], we combine mutation and crossover into a *MIP-recombination* operator applied instead of Step 3.2 in the steady-state GA. In the case of TLBP the MIP-recombination operator consists in solving a MIP problem, which is obtained from the original problem (7.3)–(7.17) as follows:

MIP-recombination operator
1. Fix all Boolean variables equal to their values in p_1.
2. Release all Boolean variables where p_1 differs from p_2 (analog of crossover).
3. Release a random subset of fixed variables independently with probability P_m (analog of mutation).

In our implementation, the MIP-solver of CPLEX 11.1 is used to find the optimum of the sub-problem emerging in the MIP-recombination. To avoid time-consuming computations we set a time limit T_{rec} for each call to the solver. Unlike the standard GA crossover, the described MIP-recombination procedure produces only one new individual at each iteration. If the parent solutions are feasible, the solver always returns a feasible solution to the MIP-

recombination sub-problem because the genotype of one of the parents is sent to CPLEX as a MIP-start feasible solution.

The initial value of mutation parameter P_m is set to P_m^0 and adapted in the process of GA execution. Every time the MIP-recombination sub-problem is solved to optimality, the parameter P_m is multiplied by 1.1. Whenever the solver is unable to find an optimal solution within the time limit T_{rec}, and $P_m > P_m^0$, parameter P_m is divided by 1.1. This adaptive mechanism keeps the complexity of the MIP-recombination problems close to the limit where the solver is able to obtain exact solutions within the given time T_{rec}.

7.5 Experimental Results

In this section, we compare the genetic algorithm and MIP-based GRASP proposed in this chapter (further referred to as GA and MIP-GRASP, respectively) to the following three heuristic and exact methods:

- the multi-start hybrid decomposition algorithm (henceforth denoted by HD) combining DAASS [10, 12] with the FSIC heuristic, where FSIC is used for construction of a feasible solution;
- the exact graph-based shortest path method [6, 10], denoted by SP;
- CPLEX 11.1 MIP-solver applied to problem (7.3)–(7.17) with the default solver settings, denoted below by CPLEX.

7.5.1 Problem Instances

In the experiments, we use five test series S1-S5 from [10], each one containing 50 randomly generated instances and two new series S6 and S7, also randomly generated but with more realistic data sets. Both series S6 and S7 consist of 20 instances. The number of operations and the upper bound m_0 on the number of stations are shown in Table 7.1. Also, this table gives the precedence constraints density, measured by the order strength (OS) of graph G. Here, by order strength we mean the number of edges in the transitive closure of graph G divided by $|\mathbf{N}|(|\mathbf{N}|-1)/2$. In series S1-S5 we have $C_1 = 10, C_2 = 2$, $\tau^b = \tau^S = 0$, $n_0 = 4$. In S6 and S7, $C_1 = 1, C_2 = 0.5$, $\tau^b = 0.2, \tau^S = 0.4$, $n_0 = 4$.

The details on the random generation of series S1-S5 can be found in [10]. The series S6 and S7 consist of more realistic instances for two reasons. Firstly, they contain non-trivial input data for parameters λ_j, s_j, while series S1-S5 in effect consist of problems in simplified formulation defined in terms of operation times t_j. Secondly, in S6 and S7 the pseudo-random choice of operations was not performed independently and uniformly as in Series S1-S5, but based on real-life data of typical shapes of the parts manufactured in

mechanical transfer lines. The random choice is applied to the shape of parts, which further defines the parameters and mutual compatibility of operations. The input data of the benchmarks in GAMS format can be found in the Discrete Location Problems Benchmarks Library by `http://www.math.nsc.ru/AP/benchmarks/english.html`.

7.5.2 Experimental Settings

The experiments were carried out on a Pentium-IV computer (3 GHz, 2.5 Gb RAM). Both the GA and the MIP-GRASP were programmed in GAMS 22.8, the rest of the algorithms being considered were coded in C++.

The tunable parameters of the constructive heuristic in MIP-GRASP and in the GA initialization were chosen as follows: $\alpha = 0.25$, $\beta = 10$. This tuning was based on preliminary experiments with different values of α and β on series S5. The frequency of finding best-known objective as a function for different values of α and β in these trials can be seen on Figure 7.1. The 95% confidence intervals for probability of finding the best-known objective value are displayed for $\beta = 20$. It follows from this figure that $\beta = 20$ is preferable for series S5. We chose $\beta = 10$ for all subsequent experiments only because on large problems of series S6 and S7 usage of $\beta = 20$ leads to so hard sub-problems, that MIP-GRASP is sometimes unable to find a feasible solution in the given amount of time.

On the basis of similar considerations, the parameters of the improvement heuristic in MIP-GRASP were set to $k_{max} = 15$, $n_{max} = 50$.

We also found that the value of the penalty term C_3 has a statistically significant impact ($p < 0.05$) on the quality of MIP-GRASP results. In order to choose C_3 adaptively while solving a given instance in GA or MIP-GRASP, we set it initially to such a small value that it does not change the optimal line design (by solving two supplementary MIP problems). Further, parameter C_3 is optimized by a simple one-dimensional search routine. This is done in MIP-GRASP restarts or in the construction of the initial population for the GA, using the average quality of solutions of the constructive heuristic as the optimization criterion.

The CPLEX tolerance parameter `optca` in the greedy heuristic was set to 2; parameter `mipstart` was set to 1 in the improvement heuristic and in

Table 7.1 Testing series

Series	1	2	3	4	5	6	7
$\|\mathbf{N}\|$	25	25	50	50	100	46 - 92	94 - 125
OS	0.5	0.15	0.9	0.45	0.25	-	-
m_0	15	4	10	15	15	23 - 46	43 - 62

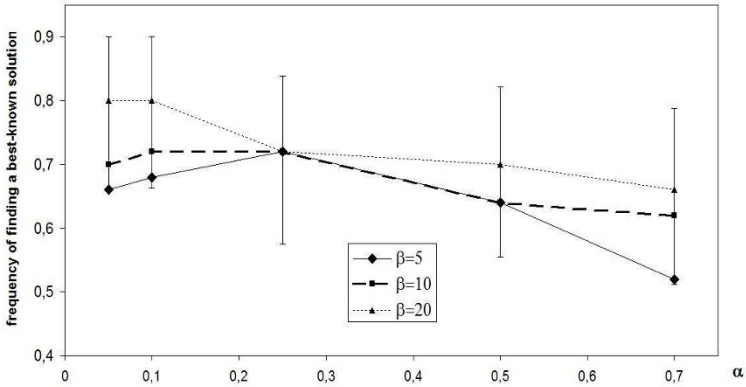


Fig. 7.1 Frequency of finding solutions with best-known objective value as a function of parameter α for different values of β in MIP-GRASP on series S5.

the MIP recombination operator, the rest of CPLEX options were set as the default. The termination conditions of the main loop in MIP-GRASP and in the GA were triggered by reaching the given CPU time limit (see the tables below).

In the experiments with the GA, we set the tournament size $s = 5$, and the initial mutation probability $P_m^0 = 0.1$. The time given for each call to the MIP-recombination operator was $T_{rec} = 5$ sec. The population size N_{ind} was set to 20 for all problems.

The GA restarting rule, described in Section 7.4, has been compared to straightforward running GA for the whole period T without restarts. The comparisons were carried out on the sets S3, S5 and S6, using the algorithmic settings described in Subsection 7.5.3 below. The GA with restarts obtained the best-known solutions more frequently on all three series. Besides that, on series S5, its advantage was shown to be statistically significant with level $p < 0.02$ in Wilcoxon matched pairs test. This motivated our choice of the restarting rule.

7.5.3 Results

In the presentation of the obtained results, we use the following notation: NS is the number of instances for which feasible solutions were found; NO and NB are the number of instances where the optimal and the best-known solutions were obtained, respectively; Δ_{max}, Δ_{avg}, Δ_{min} are the percentage of maximal, average and minimal deviation of the cost of solution from the optimal or the best-known objective value, respectively; T_{max}, T_{av}, T_{min} are

Table 7.2 Results for series S1 and S2

	Series 1				Series 2			
	SP	CPLEX	HD	MIP-GRASP	SP	CPLEX	HD	MIP-GRASP
NS	50	50	50	50	50	50	50	50
NO	**50**	**50**	39	49	**50**	**50**	35	**50**
Δ_{max}	**0**	**0**	5.26	4.16	**0**	**0**	11.1	**0**
Δ_{av}	**0**	**0**	1.0	0.08	**0**	**0**	1.7	**0**
Δ_{min}	**0**	**0**	0.0	0.0	**0**	**0**	**0**	**0**
T_{max}	**11**	841	90	90	1638	**3.53**	90	90
T_{av}	**1.4**	38	90	90	292	**0.86**	90	90
T_{min}	**0.03**	0.39	90	90	3.77	0.06	90	90

the maximal, average and minimal running time. T'_{av} is the average time till the final solution was found. Symbol "-" stands for unavailable data. The best result of a series is emphasized in bold.

7.5.3.1 Small Sized Instances

For the first two series, the available computational time was limited to 1800 seconds for the exact methods and to 90 seconds for the heuristics. The results for series S1 and S2 are reported in Table 7.2. In both series the two exact algorithms found the optima for all instances. One can see that the shortest path method performs best on series S1 in terms of computational time.

For series S2, CPLEX is the best in terms of computational time. The MIP-GRASP found all optima, outperforming HD, but these heuristics were given more CPU time than CPLEX used. In fact the average time of finding the optimal solution for MIP-GRASP was also greater than the CPLEX time. The precision of MIP-GRASP is higher on this series than on series S1, which is due to better performance of CPLEX solver on problems with low order strength (see e.g. [10]). The GA was not tested on series S1 and S2 since they are simple even for MIP-GRASP.

7.5.3.2 Medium Sized Instances

For solving the medium sized instances of series S3 and S4, the available computational time was limited to 1800 seconds for the exact methods and to 300 seconds for the heuristics. The results for series S3 and S4 are reported in Tables 7.3 and 7.4.

On series S3, the shortest path method is again the best one both in terms of the computation time and the quality of provided solutions. CPLEX found the optimal solutions in all cases and MIP-GRASP missed the optimum twice. The average and maximal time of finding the final solution for MIP-GRASP

Table 7.3 Results for series S3

	SP	CPLEX	HD	MIP-GRASP	GA
NS	50	50	50	50	50
NO	50	50	28	48	49
Δ_{max}	**0**	**0**	4.2	2.27	1.72
Δ_{av}	**0**	**0**	1.0	0.09	0.03
Δ_{min}	**0**	**0**	0	0	0
T_{max}	**0.09**	463.4	300	300	300
T_{av}	**0.04**	41.1	300	300	300
T_{min}	**0.01**	0.1	300	300	300

Table 7.4 Results for series S4

	SP	CPLEX	HD	MIP-GRASP	GA
NS	14	20	**50**	**50**	**50**
NO	14	13	39	42	**48**
Δ_{max}	-	-	13.15	13.15	**2.7**
Δ_{av}	-	-	0.86	0.64	**0.11**
Δ_{min}	-	-	**0**	**0**	**0**
T_{max}	1800	1800	300	300	300
T_{av}	1490.3	1519.0	300	300	300
T_{min}	13.0	31.5	300	300	300

was 5.02 seconds and 66 seconds, respectively, which is much shorter than the given running time. The GA demonstrated a similar behavior, slightly outperforming MIP-GRASP. The results of the hybrid method HD are inferior to those of all other algorithms in terms of the solution quality.

In series S4, the exact algorithms found feasible solutions in less than half of the cases; see Table 7.4. The CPU times were similar for these methods. In contrast, the heuristics were able to find feasible solutions in all cases, given a six times shorter computation time. The overall quality of solutions is better in the case of the GA. In general, this table shows that in cases of low density of constraints even for the medium size problems the heuristic methods are preferable, when the computational time is limited.

7.5.3.3 Large Sized Instances

The results for series S5 are reported in Table 7.5. The available computational time was limited to 5400 seconds for the exact methods and to 600 seconds for the heuristics. Both exact algorithms found less than 10 solutions; therefore, they are excluded from the table. The heuristics found feasible solutions in all 50 cases. In this series MIP-GRASP and GA demonstrate similar behavior and it is hard to tell which one is better, while HD is definitely inferior.

Table 7.5 Results for series S5

	HD	MIP-GRASP	GA
NB	11	**37**	36
Δ_{max}	35.9	30.8	**12.8**
Δ_{av}	5.02	1.6	**1.5**
T'_{av}	-	**93.4**	102.9

Table 7.6 Results for series S6 and S7

	Series 6			Series 7		
	HD	MIP-GRASP	GA	HD	MIP-GRASP	GA
NB	10	8	**15**	6	2	**17**
Δ_{max}	7.6	5.55	**4.35**	6	5.26	**1.28**
Δ_{av}	1.76	1.59	**0.72**	2.26	1.87	**0.16**
T'_{av}	-	360.4	423.57	-	1603	1323.5

Table 7.6 contains the results for series S6 and S7 with a more complex formulation based on Equation (7.2). The stand-alone CPLEX MIP solver was able to solve these problems only in several cases within 5400 seconds, so the exact results are not displayed. The computational time given to the heuristics was 900 seconds for S6 and 3000 seconds for S7. This table indicates that HD is performing similar to MIP-GRASP for what concerns the average and worst case solution quality, but HD outperforms MIP-GRASP concerning the number of best-known solutions it has found.

The GA, however, tends to outperform significantly the two other heuristics on both series S6 and S7. Most likely, this advantage of the GA is due to the effect of the MIP-recombination which constitutes the main difference of the GA from MIP-GRASP. In order to evaluate the significance of combining traits of the parent solutions in the MIP-recombination, we modified the GA so that the full MIP-recombination is performed only with a given probability P_r, otherwise we skip Step 2 in the MIP-recombination routine (this corresponds to mutation without crossover). In Figure 7.2 one can see the frequency of finding the best seen solution as a function of probability P_r for series S3, S5, S6 and S7. This figure indicates that except for the simplest set of instances S3, the capability to combine the features of both parents in a best possible way provides better output of the algorithm. This observation is also supported by the Wilcoxon matched pairs test ($p < 0.1$ for S5, $p < 0.05$ for S6).

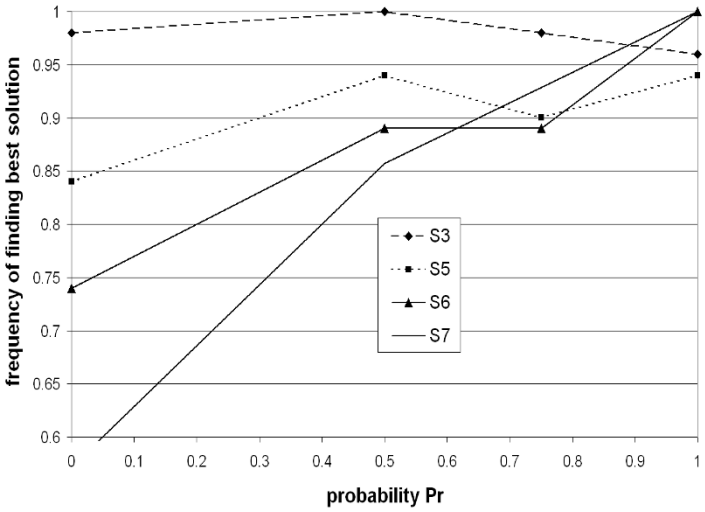


Fig. 7.2 Frequency of finding solutions with best-known objective values as a function of probability P_r.

7.6 Conclusions

In our study, firstly a GRASP approach has been developed in the MIP framework for balancing the transfer lines with multi-spindle transfer machines. The solution construction and local improvement procedures have been implemented in the GAMS environment and the results of computational experiments indicate that the method obtained is quite competitive, especially, for large-scale instances. It is important that due to the flexibility of the MIP modelling tools and robustness of the modern MIP solvers, this approach is applicable to many other large-scale problems, where the straightforward usage of branch-and-cut techniques does not yield satisfactory results.

After that, the research was aimed at the development of a more complex genetic algorithm for the TLBP, using GRASP as a supplementary heuristic for building the initial population. Although for the TLBP instances of small size this approach is not helpful, for the most difficult series of benchmarks, which are not solvable by exact methods in a reasonable amount of time, this method has a significant advantage.

The MIP-based recombination operator is shown to be useful in the genetic algorithm. In view of the wide applicability of general-purpose MIP solvers, we expect that the MIP-recombination approach may be successfully applied for many other problems (see e.g. [3]). However, in the cases where the optimal

recombination can be carried out by polynomial-time algorithms [7], these algorithms should be preferable to general purpose MIP solvers.

In subsequent research it would be valuable to compare the MIP-based approach for GRASP and GA with alternative approaches to metaheuristics, which use the shortest path method in a supplementary graph. Also, it might be helpful to implement grouping of operations into macro-operations in the improvement phase of the MIP-based GRASP. Questions of parameters tuning should be considered as well.

Acknowledgements The authors thank Michael R. Bussieck for helpful discussions on better usage of GAMS potential. The research is supported by the Russian Foundation for Basic Research, grant 07-01-00410.

References

1. I. Baybars. A survey of exact algorithms for the simple assembly line balancing. *Management Science*, 32:909–932, 1986.
2. C. Becker and A. Scholl. A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168:694–715, 2006.
3. P. Borisovsky, A. Dolgui, and A. Eremeev. Genetic algorithms for a supply management problem: MIP-recombination vs. greedy decoder. *European Journal of Operational Research*, 195:770–779, 2009.
4. A. Dolgui, B. Finel, F. Vernadat, N. Guschinsky, and G. Levin. A heuristic approach for transfer lines balancing. *Journal of Intelligent Manufacturing*, 16:159–172, 2005.
5. A. Dolgui, B. Finel, N. Guschinsky, G. Levin, and F. Vernadat. MIP approach to balancing transfer lines with blocks of parallel operations. *IIE Transactions*, 38:869–882, 2006.
6. A. Dolgui, N. Guschinsky, and G. Levin. A special case of transfer lines balancing by graph approach. *European Journal of Operational Research*, 168:732–746, 2006.
7. A. Eremeev. On complexity of optimal recombination for binary representations of solutions. *Evolutionary Computation*, 16:127–147, 2008.
8. P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys on metaheuristics*, pages 325–367. Kluwer, Boston, 2001.
9. S. Ghosh and R. Gagnon. A comprehensive literature review and analysis of the design, balancing and scheduling of assembly lines. *International Journal of Production Research*, 27(4):637–670, 1989.
10. O. Guschinskaya and A. Dolgui. A comparative evaluation of exact and heuristic methods for transfer lines balancing problem. In A. Dolgui, G. Morel, and C. Pereira, editors, *Information Control Problems in Manufacturing 2006: A Proceedings volume from the 12th IFAC International Symposium*, volume 2, pages 395–400. Elsevier, 2006.
11. O. Guschinskaya and A. Dolgui. Balancing transfer lines with multiple-spindle machines using GRASP. Unpublished manuscript, 2007.
12. O. Guschinskaya, A. Dolgui, N. Guschinsky, and G. Levin. A heuristic multi-start decomposition approach for optimal design of serial machining lines. *European Journal of Operational Research*, 189:902–913, 2008.

13. J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.
14. J. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.
15. C.R. Reeves. Genetic algorithms for the operations researcher. *INFORMS Journal on Computing*, 9(3):231–250, 1997.
16. M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003.
17. A. Scholl. *Balancing and sequencing of assembly lines.* Physica, Heidelberg, 1999.

Chapter 8
(Meta-)Heuristic Separation of Jump Cuts in a Branch&Cut Approach for the Bounded Diameter Minimum Spanning Tree Problem

Martin Gruber and Günther R. Raidl

Abstract The bounded diameter minimum spanning tree problem is an NP-hard combinatorial optimization problem arising, for example, in network design when quality of service is of concern. We solve a strong integer linear programming formulation based on so-called jump inequalities by a Branch&Cut algorithm. As the separation subproblem of identifying currently violated jump inequalities is difficult, we approach it heuristically by two alternative construction heuristics, local search, and optionally tabu search. We also introduce a new type of cuts, the center connection cuts, to strengthen the formulation in the more difficult to solve odd diameter case. In addition, primal heuristics are used to compute initial solutions and to locally improve incumbent solutions identified during Branch&Cut. The overall algorithm performs excellently, and we were able to obtain proven optimal solutions for some test instances that were too large to be solved so far.

8.1 Introduction

The *bounded diameter minimum spanning tree* (BDMST) problem is a combinatorial optimization problem appearing in applications such as wire-based communication network design when quality of service is of concern and, e.g., a signal between any two nodes in the network should not pass more than a fixed number of routers. It also arises in ad-hoc wireless networks [1] and in the areas of data compression and distributed mutual exclusion algorithms [19, 2].

Martin Gruber · Günther R. Raidl
Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria
e-mail: {gruber,raidl}@ads.tuwien.ac.at

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_8,

The goal is to identify a tree structure of minimum cost connecting all nodes of a network where the number of links between any two nodes is limited by a maximum diameter D. More formally, we are given an undirected connected graph $G = (V, E)$ with node set V and edge set E and associated costs $c_e \geq 0$, $\forall e \in E$. We seek a spanning tree $T = (V, E_T)$ with edge set $E_T \subseteq E$ whose diameter does not exceed D, where $D \geq 2$, and whose total cost $\sum_{e \in E_T} c_e$ is minimal. This problem is known to be NP-hard for $4 \leq D < |V| - 1$ [7].

8.2 Previous Work

The algorithms already published for this problem range from greedy construction heuristics, e.g. [14, 20], to various exact *(mixed) integer linear programming* (ILP) approaches. The latter include formulations based on Miller-Tucker-Zemlin inequalities [6], a compact Branch&Cut approach strengthened by connection and cycle elimination cuts [11], and in particular hop-indexed multi-commodity network flow models [8, 9] whose linear programming (LP) relaxations yield tight bounds but which involve a huge number of variables. Recently, a constraint programming approach has been proposed in [16]. Due to the complexity of the problem, exact algorithms are limited to relatively small instances with considerably less than 100 nodes when dealing with complete graphs. For larger instances, metaheuristics have been designed, e.g., evolutionary algorithms [18, 20] and a variable neighborhood search (VNS) [12]. The so far leading metaheuristics to address instances up to 1000 nodes are to our knowledge the evolutionary algorithm and the ant colony optimization algorithm from [13], which are based on a special level encoding of solutions and strong local improvement procedures.

Strongly related to the BDMST problem is the *hop constrained minimum spanning tree* (HCMST) problem, in which a root node is specified and the number of edges (hops) on each path from the root to some other node must not exceed a limit H. An overview on several ILP models and solution approaches for this problem can be found in [5]. A well working approach in particular for smaller H is the reformulation of the problem as a Steiner tree problem on a layered graph [10]. Another strong formulation is based on so-called *jump inequalities* [4]. Unfortunately, their number grows exponentially with $|V|$, and the problem of separating them in a cutting plane algorithm is conjectured to be NP-hard. Therefore, Dahl et al. [4] exploited them in a Relax&Cut algorithm where violated jump inequalities only need to be identified for integer solutions, which is straightforward.

In this work, we adopt the concept of jump inequalities to formulate a strong model for the BDMST problem, which we then solve by Branch&Cut. A hierarchy of two alternative construction heuristics, local search, and tabu search is used for efficiently separating jump cuts.

8.3 The Jump Model

Our ILP model is defined on a directed graph $G^+ = (V^+, A^+)$, with the arc set A^+ being derived from E by including for each undirected edge $(u, v) \in E$ two oppositely directed arcs (u, v) and (v, u) with the same costs $c_{u,v} = c_{v,u}$. In addition, we introduce an artificial root node r that is connected to every other node with zero costs, i.e. $V^+ = V \cup \{r\}$ and $\{(r, v) \mid v \in V\} \subset A^+$. This artificial root allows us to model the BDMST problem as a special directed outgoing HCMST problem on G^+ with root r, hop limit (i.e., maximum height) $H = \lfloor \frac{D}{2} \rfloor + 1$, and the additional constraint that the artificial root must have exactly one outgoing arc in the case of even diameter D and two outgoing arcs in the case D is odd. From a feasible HCMST $T^+ = (V^+, A_T^+)$, the associated BDMST T on G is derived by choosing all edges for which a corresponding arc is contained in A_T^+. In the odd diameter case, an additional *center edge* connecting the two nodes adjacent to the artificial root is further included.

We make use of the following variables: Arc variables $x_{u,v} \in \{0, 1\}$, $\forall (u, v) \in A^+$, which are set to one iff $(u, v) \in T^+$, and center edge variables $z_{u,v} \in \{0, 1\}$, $\forall (u, v) \in E$, which are only relevant for the odd diameter case and are set to one iff (u, v) forms the center of the BDMST.

The even diameter case is formulated as follows:

$$\text{minimize} \quad \sum_{(u,v)\in A} c_{u,v} \cdot x_{u,v} \tag{8.1}$$

$$\text{subject to} \quad \sum_{u|(u,v)\in A^+} x_{u,v} = 1 \qquad \forall\, v \in V \tag{8.2}$$

$$\sum_{v\in V} x_{r,v} = 1 \tag{8.3}$$

$$\sum_{(u,v)\in\delta^+(V')} x_{u,v} \geq 1 \qquad \forall\, V' \subset V^+ \mid r \in V' \tag{8.4}$$

$$\sum_{(u,v)\in J(P)} x_{u,v} \geq 1 \qquad \forall\, P \in P(V^+) \mid r \in S_0. \tag{8.5}$$

The objective is to minimize the total costs of all selected arcs (8.1). All nodes of the original graph (without artificial root node r) have exactly one predecessor (8.2), and just one node is successor of r (8.3). To achieve a connected, cycle free solution we include the widely used directed connection cuts (8.4), where $\delta^+(V')$ denotes all arcs (u, v) with $u \in V'$ and $v \in V^+ \setminus V'$, see also [15].

The diameter restriction is enforced by the jump inequalities (8.5) from [4] as follows. Consider a partitioning P of V^+ into $H + 2$ pairwise disjoint nonempty sets S_0 to S_{H+1} with $S_0 = \{r\}$. Let $\sigma(v)$ denote the index of the partition a node v is assigned to. Jump $J(P)$ is defined as the set of arcs

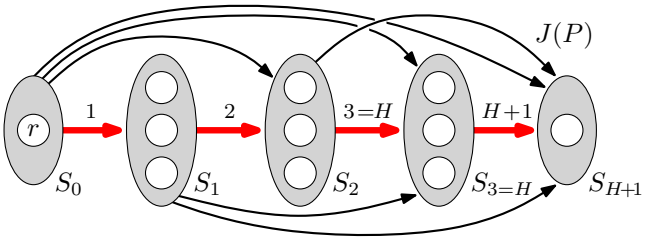


Fig. 8.1 Partitioning P of the nodes in V^+ into $H+2$ nonempty sets $S_0, \ldots, S_{H+1}$. The jump $J(P)$ contains all arcs leading from a partition to a higher indexed one skipping at least one in-between (curved arcs). A path connecting the artificial root r with nodes in S_{H+1} without any arc from $J(P)$ would consist of at least $H+1$ arcs and thus violate the hop constraint H.

$(u,v) \in A^+$ with $\sigma(u) < \sigma(v) - 1$, i.e., $J(P)$ contains all arcs leading from a partition to a higher indexed one and skipping at least one in-between, see Fig. 8.1. The jump inequality associated with this partitioning states that in a feasible HCMST T^+ at least one of these arcs in $J(P)$ must appear. Otherwise, there would be a path connecting the root contained in S_0 to a node in S_{H+1} with length at least $H+1$ violating the hop constraint. Such jump inequalities must hold for all possible partitions $P(V^+)$ of V^+ with r being element of set S_0.

The odd diameter case additionally makes use of the center edge variables $z_{u,v}$:

$$\text{minimize} \qquad \sum_{(u,v)\in A} c_{u,v} \cdot x_{u,v} + \sum_{(u,v)\in E} c_{u,v} \cdot z_{u,v} \tag{8.6}$$

$$\text{subject to} \qquad \sum_{v\in V} x_{r,v} = 2 \tag{8.7}$$

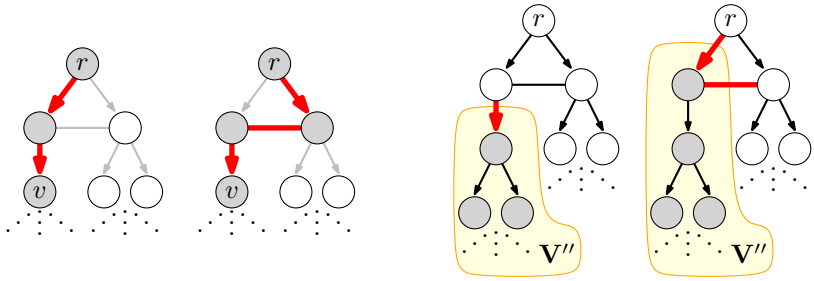
$$\sum_{v|(u,v)\in E} z_{u,v} = x_{r,u} \qquad \forall\, u \in V \tag{8.8}$$

$$2 \cdot \sum_{(u,v)\in\delta^+(V\setminus V'')} x_{u,v} + \sum_{v\in V''} x_{r,v} + \sum_{(u,v)\in\delta(V'')} z_{u,v} \geq 2 \qquad \forall\, \emptyset \neq V'' \subset V \tag{8.9}$$

(8.2), (8.4), and (8.5) are adopted unchanged.

Now, two nodes are to be connected to the artificial root node r (8.7), and they are interlinked via the center edge (8.8). The cost of this edge is also accounted for in the extended objective function (8.6).

The new connection inequalities (8.9), which we call *center connection inequalities*, are not necessary for the validity of the model but strengthen it considerably. They are essentially derived from observations in [9]: The HCMST T^+ together with the center edge linking the two center nodes connected to r forms a special structure, a so-called *triangle tree*. In such a tree



(a) Two different paths from r to v. (b) Center connection inequalities.

Fig. 8.2 Triangle tree: In the odd diameter case there are two paths connecting r with any node $v \in V$. This leads to the center connection inequalities involving the center edge.

every node $v \in V$ can be reached from r by two different – not necessarily completely arc disjoint – directed paths: The first path directly connects r with v via one center node, whereas the second one visits the second center node first and crosses the center edge, see Fig. 8.2. This idea is captured in these inequalities: Two paths from r have to reach each subset V'' of nodes of V, either from other non-center nodes (first term) or – in case a center node v is contained in V'' – directly from r and via the center edge (second and third terms).

As there are exponentially many directed and center connection inequalities (8.4, 8.9) and jump inequalities (8.5), directly solving these models is not a practical option. Instead, we start without these inequalities and apply Branch&Cut, thus, separating inequalities that are violated by optimal LP solutions on the fly. Directed connection cuts – including our special variants (8.9) – can efficiently be separated: In each LP solution $|V|$ max-flow/min-cut computations have to be performed between the artificial root r and any node of the instance graph. To compute these maximum flows in a directed graph we used the algorithm by Cherkassky and Goldberg [3]. Unfortunately, solving the separation problem for the jump inequalities is conjectured to be NP-hard [4].

8.4 Jump Cut Separation

In order to find a valid jump cut, we have to identify a node partitioning P and corresponding jump $J(P)$ for which the current LP solution $(x^{\mathrm{LP}}, z^{\mathrm{LP}})$ violates $\sum_{(u,v)\in J(P)} x^{\mathrm{LP}}_{u,v} \geq 1$.

8.4.1 Exact Separation Model

In a first attempt we formulate the separation problem as an ILP, making use of the following variables: $y_{v,i} \in \{0,1\}$, $\forall v \in V^+,\ i = 0,\ldots,H+1$, is set to one iff node v is assigned to partition S_i, and $x_{u,v} \in \{0,1\}$, $\forall (u,v) \in A^{\mathrm{LP}}$ is set to one iff arc (u,v) is contained in the jump $J(P)$; let $A^{\mathrm{LP}} = \{(u,v) \in A^+ \mid x^{\mathrm{LP}}_{u,v} > 0\}$. This leads to the following model:

$$\text{minimize} \quad \sum_{(u,v)\in A^{\mathrm{LP}}} x^{\mathrm{LP}}_{u,v} \cdot x_{u,v} \tag{8.10}$$

$$\text{subject to} \quad \sum_{i=1}^{H+1} y_{v,i} = 1 \qquad \forall\ v \in V \tag{8.11}$$

$$y_{r,0} = 1 \tag{8.12}$$

$$\sum_{v\in V} y_{v,H+1} = 1 \tag{8.13}$$

$$y_{u,i} - 1 + \sum_{j=i+2}^{H+1} y_{v,j} \le x_{u,v} \qquad \forall\ i \in \{1,\ldots,H-1\},\ (u,v) \in A^{\mathrm{LP}} \tag{8.14}$$

$$\sum_{i=2}^{H+1} y_{v,i} \le x_{r,v} \qquad \forall v \in V \mid (r,v) \in A^{\mathrm{LP}} \tag{8.15}$$

The objective is to minimize the total weight of the arcs in the jump $J(P)$ (8.10). Each node in V is assigned to exactly one of the sets S_1 to S_{H+1} (8.11), whereas the artificial root r is the only node in set S_0 (8.12). Exactly one node is assigned to set S_{H+1} (8.13), as Dahl et al. [4] showed that a jump inequality is facet-defining iff the last set is singleton. Finally, an arc (u,v) (8.14), respectively (r,v) (8.15), is part of the jump $J(P)$ iff it leads from a set S_i to a set S_j with $j \ge i+2$.

Note that, according to the following theorem, it is not necessary to explicitly address the condition that no partition may be empty:

Theorem 1. *In case all directed connection cuts are separated in advance, no partition S_i, $i \in \{1,\ldots,H\}$, will be empty in an optimal solution to the ILP model described by (8.10) to (8.15).*

Proof. Assume S_i, $i \in \{1,\ldots,H\}$, is an empty set in an otherwise valid (according to the rules defined for jump inequalities) partitioning P, $\sum_{(u,v)\in J(P)} x^{\mathrm{LP}}_{u,v} < 1$. Then V^+ can be partitioned into two sets V' and $V^+ \setminus V'$, with $V' = \{v \in V^+ \mid \sigma(v) < i\}$ (including r). The sets V' and $V^+ \setminus V'$ define a cut where all arcs from V' to $V^+ \setminus V'$ belong to the jump $J(P)$; it follows that $\sum_{(u,v)\in\delta^+(V')} x^{\mathrm{LP}}_{u,v} < 1$. Consequently, every partitioning with $\sum_{(u,v)\in J(P)} x^{\mathrm{LP}}_{u,v} < 1$ and an empty set S_i, $i \in \{1,\ldots,H\}$, can be transformed into a violated directed connection inequality, see Fig. 8.3. Since such

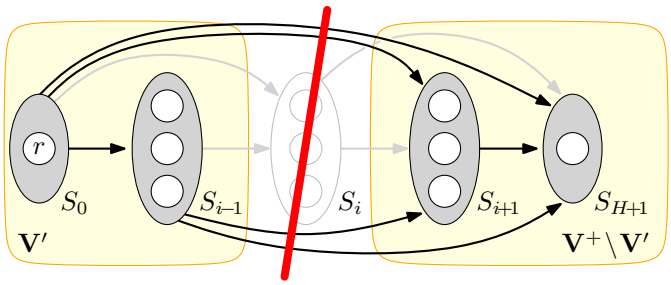


Fig. 8.3 A partitioning P with $\sum_{J(P)} x^{\mathrm{LP}} < 1$ and an empty set S_i corresponds to a violated directed connection cut.

a violated directed connection inequality does not exist in the current LP solution by assumption, no set S_i can be empty. □

This observation reveals the possibility to avoid time-consuming max-flow/min-cut computations to separate directed connection cuts. By not forcing the sets $S_1, \ldots, S_H$ to be nonempty, violated directed connection and jump constraints can be identified by only one single separation procedure, depending on whether the node partitioning P contains an empty partition S_i or not.

The exact jump cut separation model contains $O(H \cdot |V| + |A^{\mathrm{LP}}|)$ variables and $O(|V| + H \cdot |A^{\mathrm{LP}}|)$ constraints. Solving it by a general purpose solver each time when a jump cut should be separated is, however, only applicable for small problem instances as the computation times are high and increase dramatically with the problem size. According to our experiments, between about 85% and almost 100% of the total time for solving the BDMST problem is spent in this exact separation procedure for jump cuts.

To speed up computation, we developed heuristic procedures for this separation problem and apply them in the following sequence: Two alternative construction heuristics are used to find initial partitions; they are improved by local search and, in case a violated jump inequality has not yet been encountered, finally by tabu search.

8.4.2 Simple Construction Heuristic C^A

Heuristic C^A greedily assigns the nodes V^+ to sets $S_1, \ldots, S_{H+1}$ trying to keep the *number* of arcs that become part of the jump $J(P)$ as small as possible, see Algorithm 1. An independent partitioning is computed for each node $v \in V$ initially placed in the last set S_{H+1}, and the overall best solution is returned. To derive one such partitioning, all nodes u connected to r via an arc $(r, u) \in A^{\mathrm{LP}}$ with $x^{\mathrm{LP}}_{r,u}$ exceeding a certain threshold (0.5 in our experiments) are assigned to set S_1. Then the algorithm iterates through partitions S_{H+1}

Algorithm 1: Simple Construction Heuristic C^A

```
   input  : V^+, A^LP
   output: partitioning P of V^+
 1 forall nodes v ∈ V do
 2     S_0 ← {r}; S_{H+1} ← {v}; ∀i = 1,...,H : S_i ← ∅;
 3     forall arcs (r,u) | u ≠ v do
 4         if x^LP_{r,u} > 0.5 then S_1 ← S_1 ∪ {u};
 5     for i = H+1,...,3 do
 6         foreach node u ∈ S_i do
 7             foreach arc (w,u) ∈ A^LP | w not already assigned do
 8                 S_{i-1} ← S_{i-1} ∪ {w};
 9     forall still unassigned nodes u ∈ V^+ do
10         S_1 ← S_1 ∪ {u};
11     derive jump J(P) for current partitioning P = (S_0,...,S_{H+1});
12     evaluate J(P) and store P if best so far;
13 return best found partitioning;
```

down to S_3. For each of these sets S_i all arcs $(w, u) \in A^{\mathrm{LP}}$ with target node $u \in S_i$ are further examined. In case w is still *free* (i.e., not already assigned to a set), it is placed in S_{i-1}, in order to avoid (w, u) becoming part of $J(P)$. At the end, eventually remaining free nodes are assigned to set S_1.

Results achieved with heuristic C^A were encouraging, but also left room for improvement when compared to the exact separation. In particular, this heuristic does (almost) not consider differences in arc weights $x^{\mathrm{LP}}_{u,v}$ when deciding upon the assignment of nodes.

8.4.3 Constraint Graph Based Construction Heuristic C^B

To exploit arc weights in a better way, we developed the more sophisticated construction heuristic C^B which makes use of an additional *constraint graph* $G_C = (V^+, A_C)$. To avoid that an arc $(u, v) \in A^{\mathrm{LP}}$ becomes part of $J(P)$, the constraint $\sigma(u) \geq \sigma(v) - 1$ must hold in partitioning P. Heuristic C^B iterates through all arcs in A^{LP} in decreasing LP-value order (ties are broken arbitrarily) and checks for each arc whether or not its associated constraint on the partitioning can be realized, i.e., if it is compatible with previously accepted arcs and their induced constraints. Compatible arcs are accepted and collected within the constraint graph, while arcs raising contradictions w.r.t. previously accepted arcs in G_C are rejected and will be part of $J(P)$. After checking each arc in this way, a partitioning P respecting all constraints

represented by G_C is derived. Algorithm 2 shows this heuristic in pseudo-code.

In more detail, graph G_C not only holds compatible arcs but for each node $u \in V^+$ also an integer *assignment interval* $b_u = [\alpha_u, \beta_u]$ indicating the feasible range of partitions; i.e., u may be assigned to one of the sets $\{S_i \mid i = \alpha_u, \ldots, \beta_u\}$. When an arc (u, v) is inserted into A_C, the implied new constraint $\sigma(u) \geq \sigma(v) - 1$ makes the following interval updates necessary:

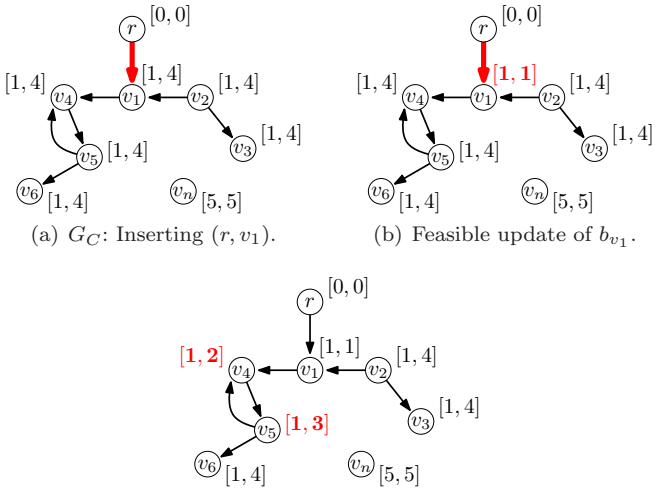
$$b_u \leftarrow [\max(\alpha_u, \alpha_v - 1),\ \beta_u] \quad \text{and} \quad b_v \leftarrow [\alpha_v,\ \min(\beta_v, \beta_u + 1)]. \tag{8.16}$$

Changes of interval bounds must further be propagated through the constraint graph by recursively following adjacent arcs until all bounds are feasible again w.r.t. the constraints.

Figure 8.4 gives an example of such an update procedure after inserting an arc into the constraint graph. It visualizes the relevant part of G_C in an instance with a diameter constraint of six, including the artificial root node r assigned to S_0 ($b_r = [0, 0]$), node v_n in partition S_{H+1} ($b_{v_n} = [5, 5]$), six additional nodes v_1 to v_6 which still are allowed to be assigned to any partition S_i, $i = 1, \ldots, 4$, and already some compatible arcs. In Fig. 8.4(a) a new arc from r to v_1 should be inserted into the constraint graph. To prevent this arc to become part of the jump $J(P)$ we have to restrict the assignment interval of v_1 (r is already fixed to a single partition): If v_1 would be assigned to any partition S_i with $i \geq 2$, the arc (r, v_1) would skip at least S_1 making it a jump arc. Therefore, the upper bound β_{v_1} has to be decreased to one ($b_{v_1} = [1, \min(4, 0{+}1)]$), see Fig. 8.4(b). Now this update has to be propagated

Algorithm 2: Constraint Graph Based Construction Heuristic C^B

input : V^+, A^{LP}
output: partitioning P of V^+
sort A^{LP} according to decreasing LP values;
forall *nodes* $v \in V$ **do**
  $S_0 \leftarrow \{r\}$; $S_{H+1} \leftarrow \{v\}$; $\forall i = 1, \ldots, H:\ S_i \leftarrow \emptyset$;
  $b_r = [0, 0]$; $b_v = [H + 1, H + 1]$; $\forall w \in V \setminus \{v\}$: $b_w \leftarrow [1, H]$;
  initialize G_C: $A_C \leftarrow \emptyset$;
  initialize jump $J(P) \leftarrow \emptyset$;
  forall *arcs* $(u, v) \in A^{\mathrm{LP}}$ *according to decreasing* $x^{\mathrm{LP}}_{u,v}$ **do**
    if $A_C \cup (u, v)$ *allows for a feasible assignment of all nodes* **then**
      $A_C \leftarrow A_C \cup (u, v)$;
      perform recursive update of bounds starting at b_u and b_v;
    else
      $J(P) \leftarrow J(P) \cup (u, v)$;
  assign nodes to partitions according to the constraints in G_C;
  evaluate jump $J(P)$ and store P if best so far;
return best found partitioning;



(a) G_C: Inserting (r, v_1). (b) Feasible update of b_{v_1}.

(c) Recursive update.

Fig. 8.4 Insertion of arc (r, v_1) into the constraint graph G_C, including all necessary updates to the assignment intervals.

through the constraint graph as shown in Fig. 8.4(c). Nothing has to be done for node v_2 (and so for v_3), it still can be assigned to any of the partitions S_1 to S_4 since the arc (v_2, v_1) can no longer become part of $J(P)$ ($\sigma(v_2) \in [1, 4]$ will always be greater than or equal to $\sigma(v_1) - 1 = 1 - 1 = 0$). On the other hand, the upper interval bound of v_4 has to be set to two (to avoid that arc (v_1, v_4) skips at least partition S_2), and – analogously – β_{v_5} has to be set to three. After this recursive update procedure the constraint graph is in a valid state again, i.e., all nodes can be assigned to partitions without violating constraints implied by the collected arcs A_C.

An arc (u, v) can be feasibly added to the graph G_C without raising conflicts with any stored constraint as long as the assignment intervals b_u and b_v do not become empty, i.e., $\alpha_u \leq \beta_u \wedge \alpha_v \leq \beta_v$ must always hold. In Algorithm 2 this condition is tested in line 8, and the arc (u, v) is either accepted for A_C or added to $J(P)$, respectively.

Theorem 2. *The recursive update of the assignment interval bounds in G_C after inserting an arc (u, v) always terminates and cannot fail if it succeeded at nodes u and v.*

Proof. Let G_C be valid, i.e., it contains no contradicting constraints, and it was possible to insert arc (u, v) into the graph without obtaining empty assignment intervals for nodes u and v. Let (s, t) be any other arc $\in G_C$, implying $\alpha_s \geq \alpha_t - 1$, and $\beta_t \leq \beta_s + 1$. Now, assume that α_t was updated, i.e. increased, to α'_t, with $\alpha'_t \leq \beta_t$. If the lower bound of s must be modified, it is set to $\alpha'_s = \alpha'_t - 1$ according to the update rules. To prove that the interval at s will not become empty we have to show that $\alpha'_s \leq \beta_s$:

Algorithm 3: Local Search

```
input  : V+, A^LP, current partitioning P and implied jump J(P)
output: possibly improved partitioning P of V+
repeat
    improved ← false;
    forall arcs (u, v) ∈ J(P) do
        if moving u to S_{σ(v)−1} or v to S_{σ(u)+1} is valid and improves solution then
            perform move; update P and J(P) correspondingly;
            improved ← true;
            break;
until improved = false ;
return partitioning P;
```

$$\alpha'_s \overset{\text{(update rule)}}{=} \alpha'_t - 1 \overset{\alpha'_t \leq \beta_t}{\leq} \beta_t - 1 \overset{\beta_t \leq \beta_s + 1}{\leq} \beta_s \tag{8.17}$$

The feasibility of the upper bound propagation can be argued in an analogous way. This also proves that the recursive update procedure terminates, even when there are cycles in G_C (intervals cannot become empty, and updates increase respectively decrease lower and upper bounds by at least one). □

8.4.4 Local Search and Tabu Search

Although the construction heuristics usually find many violated jump inequalities, there is still room for improvement using local search. The neighborhood of a current partitioning P is in principle defined by moving one node to some other partition. As this neighborhood would be relatively large and costly to search, we restrict it as follows: Each arc $(u, v) \in J(P)$ induces two allowed moves to remove it from the associated jump $J(P)$: reassigning node u to set $S_{\sigma(v)-1}$ and reassigning node v to set $S_{\sigma(u)+1}$, respectively. Moves modifying S_0 or S_{H+1} are not allowed. The local search is performed in a first improvement manner until a local optimum is reached; see Algorithm 3.

In most cases, the construction heuristics followed by local search are able to identify a jump cut if one exists. In the remaining cases, we give tabu search a try to eventually detect still undiscovered violated jump inequalities. Algorithm 4 shows our tabu search procedure in pseudo-code.

The neighborhood structure as well as the valid moves are defined as in the local search, but now a best improvement strategy is applied. Having performed a movement of a node v, we file as tabu the node v in combination with its inverted direction of movement (to a lower or higher indexed set, respectively).

Algorithm 4: Tabu Search

input : V^+, A^{LP}, current partitioning P and implied jump $J(P)$
output: possibly improved partitioning P of V^+

```
1 tabu list L ← ∅;
2 repeat
3 |   search neighborhood of P for best move m considering tabu list L;
4 |   perform move m; update P and J(P) correspondingly;
5 |   file move m^{-1} in tabu list: L ← L ∪ {m^{-1}};
6 |   remove from L entries older than max(l_min, γ · |J(P)|) iterations;
7 until no new best partitioning found during the last i_max iterations ;
8 return best encountered partitioning;
```

The tabu tenure is dynamically controlled by the number of arcs in jump $J(P)$: Tabu entries older than $\max(l_{\min}, \gamma \cdot |J(P)|)$ iterations are discarded, where $l_{\min}$ and γ are strategy parameters.

We consider the following aspiration criterion: The tabu status of a move is ignored if the move leads to a new so far best node partitioning. Tabu search terminates when a predefined number $i_{\max}$ of iterations without improvement of the overall best partitioning is reached.

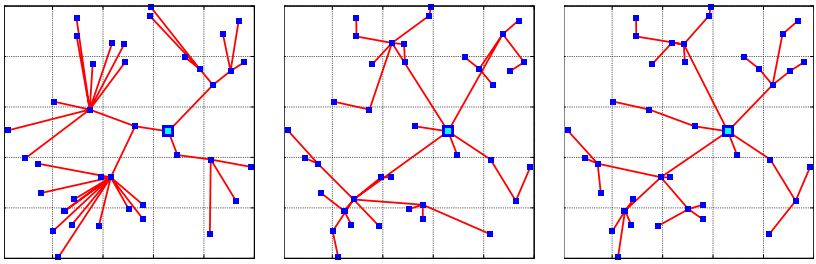
8.5 Primal Heuristics

In order to further improve the performance of our Branch&Cut approach we make use of additional fast heuristics to set an initial solution and to locally improve incumbent solutions.

In [14] Julstrom describes two different construction heuristics for the BDMST problem, the *center based tree construction* (CBTC) and the *randomized tree construction* (RTC) heuristic. Both are primarily based on Prim's MST algorithm [17] and compute, after determining a center, a height restricted tree.

CBTC simply grows a BDMST from a randomly chosen or predefined center by always adding the node with the cheapest available connection to the so long build tree without violating the height constraint. This heuristic is well suited for instances with more or less randomly generated edge weights whereas it fails miserably on Euclidean instances. The problem is that CBTC is too greedy and tends to create a backbone – the edges near the center – of extremely short edges instead of one consisting of some few but long edges spanning the whole area. As a consequence, the leaves of the BDMST have to be attached to the backbone with relatively long edges leading to a extremely poor solution as can be seen in Fig. 8.5.

To overcome this problem on Euclidean instances the RTC heuristic creates a random permutation of all nodes. The first (two) node(s) will form



(a) CBTC (8.284). (b) RTC (5.725). (c) Optimum (5.195).

Fig. 8.5 Diameter constrained trees computed by two different construction heuristics, CBTC and RTC (best solution from 100 runs), and the optimal solution (complete, Euclidean graph with 40 nodes distributed randomly in the unit square, $D = 6$). Corresponding objective values are given in parenthesis. Heuristics were forced to use the center of the optimum.

the center of the BDMST, the remaining ones are connected to the tree in the cheapest possible way in the order given by the permutation and without violating the height restriction. This approach at least increases the chance to bring longer edges into the backbone, thus leading to better final solutions.

Both construction heuristics are designed to operate on complete graphs. Whereas CBTC can handle incomplete graphs easily we modified RTC to increase the possibility of identifying a valid BDMST also on sparse graphs in the following way: Every node of the permutation not feasibly connectable is stored within a queue. After the whole permutation of nodes has been processed each node in the queue is again checked if it could be connected to the tree without violating the height restriction. This procedure is stopped when either the queue becomes empty or none of the nodes in the queue can be added feasibly to the tree. In addition, in case the diameter is odd a permutation is only accepted if the first two nodes, which should form the center, are linked via an edge.

Solutions of both construction heuristics as well as all incumbent solutions found during the optimization are further improved by the variable neighborhood descent (VND) from [13] utilizing four different neighborhood structures:

Arc exchange neighborhood: Neighboring solutions are all feasible trees that differ in exactly one arc from the current one.

Node swap neighborhood: This neighborhood contains all solutions that are obtained by exchanging the position of a node with one of its direct successors in the tree structure.

Level change neighborhood: In a neighboring solution the depth of exactly one node has been increased or decreased by one. All affected nodes are newly connected in a locally optimal way by choosing cheapest available arcs.

Center exchange level neighborhood: In neighboring solutions, the one or two center node(s) are exchanged by other nodes. The former center nodes are reconnected by cheapest possible arcs.

8.6 Computational Results

For our computational experiments we utilize Euclidean (TE) and random (TR) instances as described and used by Gouveia et al. [8, 9] as well as complete and sparse Euclidean instances of Santos et al. [6, 16]. The instance type, together with the number of nodes ($|V|$) and edges ($|E|$) and the diameter bound (D) is specified for each test case in the following results tables. All experiments have been performed on a dual-core AMD Opteron 2214 machine (2.2GHz), and CPLEX 11.1 has been used as ILP solver and framework for Branch&Cut. Since most of the heuristic components are not deterministic, the median and/or the mean value of at least 30 independent runs is listed for each experiment (when not otherwise specified). To verify statistical significance Wilcoxon rank tests with an error level of 5% (if not indicated otherwise) have been performed.

The experiments were executed with modified jump cut heuristics to simultaneously identify violated directed connection cuts to avoid additional time-consuming max-flow/min-cut computations (see proof of Theorem 1). Although a polynomial time exact separation procedure is replaced by a heuristic approach, preliminary tests demonstrated a significant enhancement in running time. Violated directed connection cuts were only identified separately in case the exact ILP model was used to separate jump cuts.

Table 8.1 demonstrates the clear advantages of applying primal heuristics: For a set of small and medium-sized instances the running times in seconds are given (heuristic jump cut separation using construction heuristic C^B with local search), as well as the mean values (including the gaps to the optimal solutions *opt*) and the standard deviations of the initial solutions. For instances with random edge costs (TR) the CBTC construction heuristic was used to compute initial solutions, RTC for all others. Since CBTC gives deterministic results for a given center it was executed once for each node $\in V$ for even diameter bounds. Otherwise, both construction heuristics were iterated until no better solution could be found for 100 runs; the finally best solution was utilized as initial solution in Branch&Cut.

The results are clear: Primal heuristics boost the optimization noticeable, especially if D is even. Significantly better results are highlighted in gray, the error probability obtained by the Wilcoxon tests is always less than 0.01%, except for instance TR 60/600/7 (0.789%). The parts of the overall running times of CBTC/RTC and the VND to improve incumbent solutions are negligibly, much less than one second for all instances. Only in some rare cases the

Table 8.1 Optimization with and without primal heuristics, running times t (in seconds), and quality of solutions, compared to the optimum *(opt)*, obtained by the construction heuristics RTC (Euclidean instances TE and Santos) or CBTC (instances with random weights TR); significantly better results according to Wilcoxon tests are highlighted gray. Since not all of the applied heuristics are not deterministic, 30 independent runs have been performed for each instance.

				t(primal heuristics)			t(no primal heuristics)			quality RTC/CBTC			
Instance	$\|V\|$	$\|E\|$	D	median	min	max	median	min	max	*opt*	mean	stddev	gap(mean)
TE	30	200	4	**11.78**	11.59	12.03	21.57	21.36	21.85	*599*	599.13	0.34	0.02%
			6	**8.92**	8.63	12.68	12.84	12.70	13.11	*482*	483.97	2.98	0.41%
			8	**1.99**	1.89	2.27	2.41	2.33	2.51	*437*	437.35	1.05	0.08%
TR	30	200	4	**1.37**	1.35	1.41	2.13	2.08	2.20	*234*	234.00	0.00	0.00%
			6	**0.61**	0.59	0.63	0.78	0.74	0.80	*157*	160.00	0.00	1.91%
			8	**0.12**	0.10	0.13	0.15	0.14	0.16	*135*	135.00	0.00	0.00%
Santos	25	300	4	**2.07**	2.02	2.12	4.06	3.98	4.12	*500*	500.00	0.00	0.00%
			6	**0.70**	0.66	0.93	1.07	1.05	1.11	*378*	378.55	1.15	0.15%
			10	**0.48**	0.40	0.56	0.59	0.55	0.62	*379*	383.06	2.13	1.07%
	40	100	4	**1.16**	1.10	1.29	1.34	1.27	1.38	*755*	759.26	11.45	0.56%
			6	**0.43**	0.40	0.45	**0.43**	0.41	0.44	*599*	621.32	2.87	3.73%
			10	**0.38**	0.36	0.41	0.39	0.37	0.41	*574*	589.42	5.58	2.69%
TE	40	400	4	**27.98**	27.18	46.24	91.98	91.23	93.60	*672*	674.32	3.35	0.35%
			6	**126.62**	93.23	243.96	182.59	181.73	189.06	*555*	558.97	1.96	0.71%
			8	**81.78**	42.37	98.84	154.92	154.01	162.29	*507*	514.94	3.05	1.57%
TR	60	600	4	**1739.10**	1647.47	1828.58	3494.98	3464.51	3645.16	*326*	368.00	0.00	12.88%
			6	**561.53**	537.10	607.79	901.11	894.57	937.41	*175*	179.00	0.00	2.29%
			8	**4.66**	4.53	4.89	4.74	4.67	4.89	*127*	148.00	0.00	16.54%
TE	30	200	5	67.50	45.67	69.34	**52.96**	52.54	53.74	*534*	534.29	0.90	0.05%
			7	28.98	24.91	31.95	**28.34**	27.92	28.91	*463*	464.68	1.58	0.36%
TR	30	200	5	2.67	2.36	3.64	**2.39**	2.35	2.44	*195*	196.52	3.11	0.78%
			7	**0.29**	0.27	0.34	0.32	0.31	0.33	*144*	145.26	3.20	0.87%
Santos	25	300	5	**10.42**	10.27	10.59	10.65	10.52	10.88	*429*	429.00	0.00	0.00%
			7	**2.13**	2.11	2.16	3.85	3.79	3.92	*408*	408.00	0.00	0.00%
			9	**1.11**	1.08	1.41	1.62	1.58	1.64	*336*	337.19	1.83	0.36%
	40	100	5	**0.93**	0.87	1.02	1.06	1.02	1.10	*729*	739.35	14.37	1.42%
			7	**3.38**	2.90	4.30	4.52	4.47	4.65	*667*	684.87	7.12	2.68%
			9	**3.44**	3.30	3.81	3.95	3.90	4.05	*552*	570.77	8.79	3.40%
TE	40	400	5	**348.51**	335.09	618.57	466.34	464.20	478.88	*612*	613.55	2.41	0.25%
			7	**463.89**	244.64	808.79	605.31	601.90	623.02	*527*	532.84	3.38	1.11%
			9	**181.40**	111.62	822.45	527.47	524.99	544.38	*495*	502.74	3.68	1.56%
TR	60	600	5	1286.76	652.53	2546.96	**811.16**	804.56	835.89	*256*	265.71	11.09	3.79%
			7	33.37	17.44	52.10	**27.31**	27.01	28.06	*150*	163.35	3.90	8.90%
			9	**5.99**	5.33	20.88	10.32	10.17	10.62	*124*	136.35	2.74	9.96%

primal heuristics can mislead CPLEX, although the minimal running times achieved are still better or at least comparable.

The solutions computed by CBTC and RTC for these small instances are in general of high quality (average objective value less than 2% from the optimum) when the graph is complete or at least dense. On sparse graphs (Santos 40/100, TR 60/600) already finding a feasible solution is difficult. An interesting observation is that the running times are much more stable when no primal heuristics are used, so differences in the jump cuts identified by C^B plus local search have only a relatively small impact in this case. For all remaining experiments primal heuristics were activated.

For smaller instances where the exact ILP-based jump cut separation can also be applied, Table 8.2 lists success rates SR(·) for finding existing violated jump inequalities in LP solutions for the two construction heuristics (C^A and

Table 8.2 Success rates SR (%) for separating jump cuts by construction heuristics C^A and C^B, optionally followed by local search L and tabu search T, in comparison to the exact separation approach on the same LP solutions.

Instance	$\|V\|$	$\|E\|$	D	#exact	SR(C^A)	SR(C^AL)	SR(C^B)	SR(C^BL)	SR(C^BLT)
TE	30	200	4	817	99.02%	**100.00%**	99.14%	99.39%	99.39%
			6	991	97.17%	**99.80%**	97.07%	97.58%	98.63%
			8	560	65.87%	92.94%	95.08%	95.42%	**96.35%**
TR	30	200	4	272	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			6	152	98.03%	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			8	22	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
Santos	25	300	4	316	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			6	126	99.21%	99.21%	**100.00%**	**100.00%**	**100.00%**
			10	77	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
	40	100	4	204	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			6	112	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			10	85	64.71%	90.59%	**96.47%**	**96.47%**	**96.47%**
TE	30	200	5	2786	89.75%	**98.39%**	92.41%	95.36%	95.36%
			7	3353	64.04%	91.88%	94.06%	95.41%	**96.99%**
TR	30	200	5	377	79.05%	91.51%	96.55%	**97.35%**	**97.35%**
			7	89	80.90%	85.39%	92.13%	94.38%	**95.51%**
Santos	25	300	5	794	83.50%	97.10%	97.73%	98.36%	**99.46%**
			7	188	81.38%	88.83%	95.21%	95.74%	**96.81%**
			9	115	91.30%	93.91%	97.39%	97.39%	**98.26%**
	40	100	5	186	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			7	445	81.88%	93.82%	95.58%	96.15%	**96.16%**
			9	485	67.80%	73.35%	92.66%	93.04%	**94.02%**

C^B), optionally followed by local search (L) and tabu search (T) with the strategy parameters $l_{\min} = 5$, $\gamma = 0.75$, and $i_{\max} = 25$. The number of cuts identified by the exact model is given in column "#exact". As can be seen, for even diameter already the simple construction heuristic C^A gives excellent results, in most cases further improved by local search. The statistically significantly better heuristic C^B (error level $< 0.01\%$) leaves not much room for local and tabu search to enhance the success rate. A more differentiated situation can be observed for odd diameter bounds. The number of jump cuts identified directly by C^B is significantly higher in contrast to C^A (error level $< 0.01\%$), whereas local search flattens the differences in the construction phase to a greater or lesser extent. On almost all test instances, tabu search further improves the success rate to more than 95%. In total, heuristic C^B followed by local search and tabu search was able to separate all existing jump cuts for 9 out of 22 instances.

The consequences of the success to reliably identify violated jump inequalities can be seen in Table 8.3, where for the various approaches CPU-times t(·) to identify proven optimal integer solutions are listed. It can clearly be seen that the excessive running times of the exact jump cut separation prohibit its usage on larger instances. Times of the overall optimization process are in general magnitudes higher as when using our heuristics for jump cut separation, sometimes even the given CPU-time limit of one hour is exceeded. Since tabu search is only executed in case the construction heuristic followed

Table 8.3 Optimal solution values, median running times t (in seconds) to find and prove these solutions when using different strategies for jump cut separation, and optimality gaps of the final LP relaxations in the root nodes of the Branch&Cut search trees when using heuristic C^B followed by local search and tabu search. The last column gives running times in case directed connection cuts (dc) are separated exactly using multiple max-flow/min-cut computations.

Instance	$\|V\|$	$\|E\|$	D	*opt*	t(exact)	t(C^AL)	t(C^BL)	t(C^BLT)	gap(C^BLT)	t(dc+C^BLT)
TE	30	200	4	*599*	3522.73	13.03	11.78	**11.39**	1.69%	18.73
			6	*482*	> 1h	32.06	**8.92**	9.09	2.59%	13.73
			8	*437*	> 1h	2.16	**1.99**	2.12	1.98%	3.25
TR	30	200	4	*234*	328.09	1.63	**1.37**	1.38	0.00%	3.28
			6	*157*	185.65	0.96	**0.61**	0.63	0.00%	1.16
			8	*135*	0.59	**0.11**	0.12	**0.11**	0.00%	0.30
Santos	25	300	4	*500*	809.86	7.03	**2.07**	2.10	0.00%	3.58
			6	*378*	215.30	1.04	**0.70**	0.71	0.53%	0.86
			10	*379*	419.03	0.58	**0.48**	**0.48**	0.00%	0.64
	40	100	4	*755*	105.34	**0.98**	1.16	1.18	0.00%	2.14
			6	*599*	41.07	**0.37**	0.43	0.43	0.00%	0.93
			10	*574*	440.55	**0.34**	0.38	0.36	0.13%	0.70
TE	30	200	5	*534*	> 1h	**57.85**	67.50	62.14	7.20%	148.88
			7	*463*	> 1h	28.87	28.98	**28.35**	6.63%	38.16
TR	30	200	5	*195*	831.31	2.86	**2.67**	2.85	9.40%	5.36
			7	*144*	139.08	**0.27**	0.29	0.30	4.56%	1.31
Santos	25	300	5	*429*	1122.52	7.20	10.42	**6.08**	8.87%	20.08
			7	*408*	2489.67	**1.69**	2.13	1.98	4.65%	6.10
			9	*336*	66.66	**1.01**	1.11	1.12	0.89%	1.28
	40	100	5	*729*	238.24	**0.79**	0.93	1.02	0.00%	2.98
			7	*667*	988.36	**2.47**	3.38	3.22	1.50%	5.32
			9	*552*	> 1h	7.47	**3.44**	3.98	3.22%	5.70

by local search fails to identify a violated jump inequality, running times of C^BL and C^BLT considerably differ only on few instances, especially when D is odd.

On these relatively small instances it is difficult to draw conclusions on the performance of the various heuristics, even though the time required to solve all instances to proven optimality is lowest for C^B with local search and tabu search (141.02s), followed by C^BL (150.86s) and C^AL (170.77s). The picture becomes more apparent when investigating slightly larger instances (sparse, dense, and complete graphs), see Table 8.4. Again, statistically significantly better results are highlighted gray; the error probability is always less than 0.01% except for instances TE 30/435/9 (0.5%), TR 40/480/7 (2.73%; C^AL is significantly faster although median(C^BL)<median(C^AL)), TR 40/480/9 (4.17%), and TR 40/780/7 (1.72%). With increasing instance size the higher success rates of C^BL in identifying jump cuts show a considerable impact on running times.

To achieve a good runtime behavior using tabu search a lot of parameter tuning for $l_{\min}$, γ, and $i_{\max}$ is necessary. A parameter set working for all instance types and sizes very well does not exist. In addition, when the number of nodes and edges in the graph increases, the benefit of identifying more violated jump inequalities is increasingly undone. Especially this is true when

Table 8.4 Running times t (in seconds) on larger instances (sparse, dense, complete) when separating jump cuts using heuristics C^A and C^B including local search; statistically significantly better results are highlighted gray.

Instance		$\|V\|$	$\|E\|$	D	t(C^AL)	t(C^BL)	D	t(C^AL)	t(C^BL)
sparse	TE	30	175	4	9.40	**9.31**	5	112.39	**72.05**
				6	28.66	**6.62**	7	**23.07**	28.65
				8	2.09	**1.62**	9	**1.49**	**1.49**
dense			305	4	98.95	**27.08**	5	35.38	**33.51**
				6	24.01	**11.28**	7	**12.09**	27.10
				8	2.70	**2.01**	9	**1.47**	1.80
complete			435	4	98.68	**30.74**	5	54.49	**32.64**
				6	47.57	**13.18**	7	**13.00**	19.73
				8	2.68	**2.60**	9	**2.37**	2.64
sparse	TR	40	175	4	63.59	**24.27**	5	174.60	**20.03**
				6	10.28	**2.08**	7	3.82	**1.63**
				8	**0.46**	0.47	9	0.84	**0.72**
dense			480	4	173.81	**27.55**	5	24.63	**20.78**
				6	8.34	**2.71**	7	3.21	**3.09**
				8	0.77	**0.72**	9	1.15	**1.10**
complete			780	4	206.48	**27.75**	5	100.00	**68.67**
				6	7.60	**3.61**	7	**15.27**	15.50
				8	**1.08**	1.10	9	9.13	**8.96**
				sum:	787.15	**194.70**		588.40	**360.09**

D is odd since a lot of computational effort is invested into LP solutions in which no jump cuts exist. Therefore, we abstained from using tabu search on larger instances since the performance of the construction heuristics with local search is already excellent.

Table 8.3 also lists optimal solution values ("opt") as well as optimality gaps of the LP relaxations at the root nodes of the Branch&Cut search trees for C^BLT. Whereas our model is quite tight in the even diameter case, the gaps for odd diameters reveal potential for further investigations to strengthen the formulation. In the last column, Table 8.3 finally gives running times for C^BLT when directed connection cuts (dc) are separated for LP solutions before jump cuts using an exact max-flow/min-cut algorithm, which proved to be definitely much more time consuming by a factor of at least 1.2 up to 4 and more.

Last but not least, Table 8.5 compares our approach to the so far leading hop-indexed multi-commodity flow formulations from [8] (even diameter cases) and [9] (odd diameter cases) on larger instances. The columns list for each instance the optimal objective value if known, otherwise an upper bound (opt/UB*), the LP relaxation value for construction heuristic C^B with local search (LP(C^BL)), the gaps for this approach and for the best model from [8] and [9] whenever the optimum is available resp. the corresponding values were published (gap(C^BL), gap(GMR)), as well as the running time to proven optimality (t(C^BL)); a time limit of 10 hours was used for these experiments.

We were able to discover and prove previously unknown optima (bold) and could show that instance TE 80/800/4 is infeasible. Concerning the LP

Table 8.5 Optimal values resp. upper bounds, LP relaxation values, LP gaps (for C^BL and GMR, the tightest models from [8] and [9]), and running times on Euclidean and random instances with 40, 60, and 80 nodes.

								t(C^BL)		
Instance	$\|V\|$	$\|E\|$	D	*opt*/UB*	LP(C^BL)	gap(C^BL)	gap(GMR)	median	min	max
TE	40	400	4	*672*	672.00	0.00%	0.04%	27.98	27.18	46.24
			6	*555*	544.33	1.92%	0.60%	126.62	93.23	243.96
			8	*507*	500.14	1.35%	0.50%	81.78	42.37	98.84
	60	600	4	*1180*	1178.50	0.13%	0.10%	1062.03	673.11	1154.82
			6	*837*	816.85	2.41%	0.50%	9244.26	5331.65	16389.33
			8	**755**	736.60	2.44%		18844.98	15815.31	25913.07
	80	800	4	**infeasible**		infeasible		1871.81	1857.74	2098.96
			6	**1066**	1044.87	1.98%			> 10h	
			8	963*	925.32	*3.91%			> 10h	
TR	40	400	4	*309*	309.00	0.00%	0.00%	23.35	22.84	23.99
			6	*189*	189.00	0.00%	0.00%	2.82	2.78	2.90
			8	*161*	161.00	0.00%	0.00%	0.76	0.72	0.79
	60	600	4	*326*	323.49	0.77%	0.70%	1739.10	1647.47	1828.58
			6	*175*	171.16	2.19%	1.30%	561.53	537.10	607.79
			8	*127*	127.00	0.00%	0.00%	4.66	4.53	4.89
	80	800	4	*424*	399.67	5.74%	5.70%		> 10h	
			6	**210**	206.41	1.71%		1904.19	1891.74	2181.73
			8	**166**	164.33	1.00%		25.56	24.83	27.24
TE	40	400	5	*612*	578.42	5.49%	0.00%	348.51	335.09	618.57
			7	*527*	495.09	6.06%	0.30%	463.89	244.64	808.79
			9	*495*	468.08	5.44%	0.30%	181.40	111.62	822.45
	60	600	5	*965*	899.79	6.76%	0.00%	34288.91	31383.42	> 10h
			7	*789*	742.23	5.93%	0.00%		> 10h	
			9	*738*	690.88	6.38%	0.50%	> 10h	30869.08	> 10h
	80	800	5	**1313**	1205.82	8.16%			> 10h	
			7	**1010**	942.60	6.67%			> 10h	
			9	950*	871.90	*8.22%			> 10h	
TR	40	400	5	*253*	224.90	11.11%	1.00%	17.94	17.66	22.49
			7	*171*	169.11	1.10%	0.00%	2.16	2.00	2.26
			9	*154*	154.00	0.00%	0.00%	1.06	0.86	1.20
	60	600	5	*256*	217.14	15.18%	3.20%	1286.76	652.53	2546.96
			7	*150*	138.50	7.67%	0.30%	33.37	17.44	52.10
			9	*124*	119.84	3.35%	0.00%	5.99	5.33	20.88
	80	800	5	**323**	272.42	15.66%			> 10h	
			7	**185**	176.44	4.62%		153.57	126.16	300.28
			9	**158**	154.57	2.17%		15.97	13.81	133.14

gaps, the results are comparable on even diameter instances, while for odd diameters the flow models are significantly better. A fair runtime comparison to [8] and [9] is not possible since the used hardware is too different (dual-core AMD Opteron 2214 (2.2GHz) compared to an Intel Pentium II (450MHz)). A rough estimation indicates that the flow formulations have their strengths on small diameter bounds (4 to 6), whereas Branch&Cut dominates when the diameter bound is looser (6 and above). To give an example: In [9] Gouveia et al. report for their best odd diameter formulation, the Longest-Path model, on instance TE 40/400/5 a running time of 345 seconds to prove optimality, the Branch&Cut approach requires about the same time on a much faster machine (median: 348.51 seconds). On the same instance with a diameter

bound of 9 the situation changes, Gouveia et al. list 44600 seconds for their model whereas Branch&Cut in general only requires about 181.40 seconds (median).

8.7 Conclusions and Future Work

In this work we presented a new ILP formulation for the BDMST problem utilizing jump inequalities to ensure the diameter constraint and solve it with Branch&Cut. The odd diameter case is further strengthened by new center connection inequalities. For the separation of jump inequalities we considered an exact ILP approach and two greedy construction heuristics followed by local and tabu search. While our exact separation prohibits its use in practice due to its excessive computation times, the heuristic methods are substantially faster and achieve convincing success rates in identifying violated jump inequalities; they lead to an excellent overall performance of the Branch&Cut.

The usage of primal heuristics for determining initial solutions and for locally improving new incumbent solutions enhances our approach significantly. The gain received by replacing an exact polynomial time separation procedure for directed connection cuts by fast (meta-)heuristics was surprisingly high and can be an interesting field for further research also for other types of cuts and problems. Having an exact algorithm at hand to solve BDMST instances of moderate size in reasonable time also opens up new opportunities in combining it with leading metaheuristics. Smaller subproblems arising can now be solved to proven optimality, or specially designed neighborhoods can be searched making use of the Branch&Cut approach.

References

1. K. Bala, K. Petropoulos, and T.E. Stern. Multicasting in a linear lightwave network. In *Proc. of the 12th IEEE Conference on Computer Communications*, pages 1350–1358. IEEE Press, 1993.
2. A. Bookstein and S. T. Klein. Compression of correlated bit-vectors. *Information Systems*, 16(4):387–400, 1991.
3. B.V. Cherkassky and A.V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997. Code available at `http://www.avglab.com/andrew/CATS/maxflow_solvers.htm`.
4. G. Dahl, T. Flatberg, N. Foldnes, and L. Gouveia. Hop-constrained spanning trees: The jump formulation and a relax-and-cut method. Technical report, University of Oslo, Centre of Mathematics for Applications (CMA), 2005.
5. G. Dahl, L. Gouveia, and C. Requejo. On formulations and methods for the hop-constrained minimum spanning tree problem. In *Handbook of Optimization in Telecommunications*, chapter 19, pages 493–515. Springer Science + Business Media, 2006.

6. A.C. dos Santos, A. Lucena, and C.C. Ribeiro. Solving diameter constrained minimum spanning tree problems in dense graphs. In *Proceedings of the International Workshop on Experimental Algorithms*, volume 3059 of *LNCS*, pages 458–467. Springer Verlag, Berlin, 2004.
7. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.
8. L. Gouveia and T.L. Magnanti. Network flow models for designing diameter-constrained minimum spanning and Steiner trees. *Networks*, 41(3):159–173, 2003.
9. L. Gouveia, T.L. Magnanti, and C. Requejo. A 2-path approach for odd-diameter-constrained minimum spanning and Steiner trees. *Networks*, 44(4):254–265, 2004.
10. L. Gouveia, L. Simonetti, and E. Uchoa. Modelling the hop-constrained minimum spanning tree problem over a layered graph. In *Proceedings of the International Network Optimization Conference*, pages 1–6, Spa, Belgium, 2007.
11. M. Gruber and G.R. Raidl. A new 0–1 ILP approach for the bounded diameter minimum spanning tree problem. In L. Gouveia and C. Mourão, editors, *Proceedings of the International Network Optimization Conference*, volume 1, pages 178–185, Lisbon, Portugal, 2005.
12. M. Gruber and G.R. Raidl. Variable neighborhood search for the bounded diameter minimum spanning tree problem. In P. Hansen, N. Mladenović, J.A. Moreno Pérez, editors, *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search*, Tenerife, Spain, 2005.
13. M. Gruber, J. van Hemert, and G.R. Raidl. Neighborhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA, and ACO. In *Proceedings of the Genetic and Evolutionary Computation Conference 2006*, volume 2, pages 1187–1194, 2006.
14. B.A. Julstrom. Greedy heuristics for the bounded-diameter minimum spanning tree problem. Technical report, St. Cloud State University, 2004.
15. T.L. Magnanti and L.A. Wolsey. *Handbooks in Operations Research and Management Science: Network Models*, chapter 9. North-Holland, 1995.
16. T.F. Noronha, A.C. Santos, and C.C. Ribeiro. Constraint programming for the diameter constrained minimum spanning tree problem. *Electronic Notes in Discrete Mathematics*, 30:93–98, 2008.
17. R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.
18. G.R. Raidl and B.A. Julstrom. Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In G. Lamont, H. Haddad, G.A. Papadopoulos, and B. Panda, editors, *Proceedings of the ACM Symposium on Applied Computing*, pages 747–752. ACM Press, 2003.
19. K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.
20. A. Singh and A.K. Gupta. Improved heuristics for the bounded-diameter minimum spanning tree problem. *Soft Computing*, 11(10):911–921, 2007.

Chapter 9
A Good Recipe for Solving MINLPs

Leo Liberti, Giacomo Nannicini, and Nenad Mladenović

Abstract Finding good (or even just feasible) solutions for Mixed-Integer Nonlinear Programming problems independently of the specific problem structure is a very hard but practically useful task, especially when the objective and/or the constraints are nonconvex. We present a general-purpose heuristic based on Variable Neighbourhood Search, Local Branching, Sequential Quadratic Programming and Branch-and-Bound. We test the proposed approach on the MINLPLib, discussing optimality, reliability and speed.

9.1 Introduction

The mathematical programming formulation $\min\{f(x) \mid g(x) \leq 0\}$ can be ascribed to four different categories: Linear Programming (LP) if f, g are linear forms and $x \in \mathbb{R}^n$ are continuous variables, Mixed-Integer Linear Programming (MILP) if some of the variables are integer, Nonlinear Programming (NLP) if there are some nonlinear functions in f, g and the variables are continuous, Mixed-Integer Nonlinear Programming (MINLP) if f, g involve nonlinear functions and the vector x includes some integer variables; problems are also categorized according to the convexity of objective function and constraints. In general, solving LPs and convex NLPs is considered easy, and solving MILPs, nonconvex NLPs and convex MINLPs (cMINLPs) is considered difficult. Solving nonconvex MINLPs involves difficulties arising from

Leo Liberti · Giacomo Nannicini
LIX, École Polytechnique, Palaiseau, France
e-mail: {liberti,giacomon}@lix.polytechnique.fr

Nenad Mladenović
Brunel University, London, UK and Institute of Mathematics, Academy of Sciences, Belgrade, Serbia
e-mail: nenad.mladenovic@brunel.ac.uk,nenad@turing.mi.sanu.ac.yu

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_9,

both nonconvexity and integrality, and it is considered the hardest problem of all. From the modelling point of view, however, nonconvex MINLPs are the most expressive mathematical programs — it stands to reason, then, that general-purpose MINLP solvers should be very useful. Currently, optimal solutions of MINLPs in general form are obtained by using the spatial Branch-and-Bound (sBB) algorithm [2, 29, 38, 39]; but guaranteed optima can only be obtained for relatively small-sized MINLPs. Realistically-sized MINLPs can often have thousands (or tens of thousands) of variables (continuous and integer) and nonconvex constraints. With such sizes, it becomes a great challenge to even find a feasible solution, and sBB algorithms become almost useless. Some good solvers targeting cMINLPs exist in the literature [1, 4, 6, 16, 17, 27]; although they can all be used on nonconvex MINLPs as well (forsaking the optimality guarantee), in practice their mileage varies wildly with the instance of the problem being solved, resulting in a high fraction of "false negatives" (i.e. feasible problems for which no feasible solution was found). The Feasibility Pump (FP) idea was recently extended to cMINLPs [5], but again this does not work so well when applied to nonconvex MINLPs unmodified [35].

In this chapter, we propose an effective and reliable MINLP heuristic, called the Relaxed-Exact Continuous-Integer Problem Exploration (RECIPE) algorithm. The MINLPs we address are cast in the following general form:

$$\left.\begin{array}{rrcl} \min\limits_{x\in\mathbb{R}^n} & & f(x) & \\ \text{s.t.} & l \leq & g(x) & \leq u \\ & x^L \leq & x & \leq x^U \\ & x_i \in \mathbb{Z} & & \forall\, i \in Z \end{array}\right\} \tag{9.1}$$

In the above formulation, x are the decision variables (x_i is integer for each $i \in Z$ and continuous for each $i \notin Z$, where $Z \subseteq \{1, \ldots, n\}$). $f : \mathbb{R}^n \to \mathbb{R}$ is a possibly nonlinear function, $g : \mathbb{R}^n \to \mathbb{R}^m$ is a vector of m possibly nonlinear functions (assumed to be differentiable), $l, u \in \mathbb{R}^m$ are the constraint bounds (which may be set to $\pm\infty$), and $x^L, x^U \in \mathbb{R}^n$ are the variable bounds.

RECIPE puts together a global search phase based on Variable Neighbourhood Search (VNS) [22] and a local search phase based on a Branch-and-Bound (BB) type heuristic. The VNS global phase rests on neighbourhoods defined as hyperrectangles for the continuous and general integer variables [30] and by Local Branching (LB) for the binary variables [15]. The local phase employs a BB solver for convex MINLPs [17], and applies it to (possibly nonconvex) MINLPs, making therefore effectively a heuristic. A local NLP solver, which implements a Sequential Quadratic Programming (SQP) algorithm [20], supplies an initial constraint-feasible solution to be employed by the BB as starting point. RECIPE is an efficient, effective and reliable general-purpose algorithm for solving complex MINLPs of small and medium scale.

The original contribution of this chapter is the way a set of well-known and well-tested tools are combined into making a very powerful global optimization method. This chapter does not contribute theoretical knowledge but rather the description of a practically useful algorithm whose easy implementation rests on existing off-the-shelf software tools complemented by relatively few lines of code. It turns out that RECIPE, when acting on the whole MINLPLib library [9], is able to find optima equal to or better than the best solutions reported in the literature for 55% of the instances. The closest competitor is SBB+CONOPT [10, 13], which matches or surpasses the best solutions listed in MINLPLib on only 37% of the instances. We improve the best known solutions in 7% of the cases.

The rest of this chapter is organized as follows. In Section 9.2 we describe the basic component algorithms on which RECIPE is based. Section 9.3 presents the overall approach. In Section 9.4 we discuss computational results obtained over MINLPLib, focusing on optimality, reliability and speed. Section 9.5 concludes the chapter.

9.2 The Basic Ingredients

This section describes the four main components used in RECIPE, which are:

- the global search phase: Variable Neighbourhood Search;
- the binary variable neighbourhood definition technique: Local Branching;
- the constraint and integral feasibility enforcing local solution algorithm: Branch-and-Bound for cMINLPs;
- the constraint feasibility enforcing local solution algorithm: Sequential Quadratic Programming.

9.2.1 Variable Neighbourhood Search

VNS relies on iteratively exploring neighbourhoods of growing size to identify better local optima [22, 23, 24]. More precisely, VNS escapes from the current local minimum x^* by initiating other local searches from starting points sampled from a neighbourhood of x^* which increases its size iteratively until a local minimum better than the current one is found. These steps are repeated until a given termination condition is met. This can be based on CPU time, number of non-improving steps and other configurable parameters.

VNS has been applied to a wide variety of problems both from combinatorial and continuous optimization [3, 7, 12, 26, 31, 32, 37]. Its early applications to continuous problems were based on a particular problem structure. In the continuous location-allocation problem, the neighbourhoods are defined according to the meaning of problem variables (assignments of facilities

to customers, positioning of yet unassigned facilities and so on) [7]. In bilinearly constrained bilinear problems the neighbourhoods are defined in terms of the applicability of the successive linear programming approach, where the problem variables can be partitioned so that fixing the variables in either set yields a linear problem; more precisely, the neighbourhoods of size k are defined as the vertices of the LP polyhedra that are k pivots away from the current vertex [22]. The first VNS algorithm targeted at problems with fewer structural requirements, namely, box-constrained nonconvex NLPs, was given in [36] (the paper focuses on a particular class of box-constrained NLPs, but the proposed approach is general). Its implementation is described in [11]. Since the problem is assumed to be box-constrained, the neighbourhoods arise naturally as hyperrectangles of growing size centered at the current local minimum x^*. The same neighbourhoods were used in [30], an extension of VNS to constrained NLPs.

9.2.2 Local Branching

LB is an efficient heuristic for solving difficult MILP problems [15]. Given an integer $k > 0$, the LB search explores k-neighbourhoods of the incumbent x^* by allowing at most k of the integer variables to change their value; this condition is enforced by means of the *local branching constraint*:

$$\sum_{i \in \bar{S}} (1 - x_i) + \sum_{i \notin \bar{S}} x_i \leq k, \tag{9.2}$$

where $\bar{S} = \{i \leq n \mid i \in Z \wedge x_i^* = 1\}$, which defines a neighbourhood of radius k with respect to the binary variables of (9.1), centered at a binary solution with support $\bar{S}$. LB updates the incumbent as it finds better solutions. When this happens, the LB procedure is called iteratively with $\bar{S}$ relative to the new incumbent. We remark that LB was successfully used in conjunction with VNS in [25].

9.2.3 Branch-and-Bound for cMINLPs

Solving cMINLPs (i.e. MINLPs where the objective function and constraints are convex — the terminology is confusing as all MINLPs are actually nonconvex problems because of the integrality constraints) is conceptually not much more difficult than solving MILPs: as the relaxed problem is convex, obtaining lower bounds is easy. The existing tools, however, are still far from the quality attained by modern MILP solvers. The problem is usually solved by BB, where only the integer variables are selected for branching. A re-

stricted (continuous) convex NLP is formed and solved at each node, where the variable ranges have been restricted according to the node's definition. Depending on the algorithm, the lower bounding problem at each node may either be the original problem with relaxed integrality constraints [10, 17] (in which case the BB becomes a recursive search for a solution that is both integer feasible and a local optimum in continuous space), or its linear relaxation by outer approximation [1, 4, 14, 16]. In the former case, the restricted NLP is solved to optimality at each node by using local NLP methods (which converge to the node's global optimum when the problem is convex) such as SQP (see Section 9.2.4), in the latter it is solved once in a while to get good incumbent candidates.

Another approach to solving MINLPs, which can be applied to convex and pseudoconvex objective and constraints alike, is taken in [40, 41, 42], where a cutting planes approach is blended in with a sequence of MILP subproblems (which only need to be solved to feasibility).

These approaches guarantee an optimal solution if the objective and constraints are convex, but may be used as a heuristic even in presence of nonconvexity. Within this chapter, we employ these methods in order to find local optima of general (nonconvex) MINLPs. The problem of finding an initial feasible starting point (used by the BB local NLP subsolver) is addressed by supplying the method with a constraint feasible (although not integer feasible) starting point found by an SQP algorithm (see Section 9.2.4).

9.2.4 Sequential Quadratic Programming

SQP methods find local solutions to nonconvex NLPs. They solve a sequence of quadratic approximations of the original problem subject to a linearization of its constraints. The quadratic approximation is obtained by a convex model of the objective function Hessian at a current solution point, subject to a linearization of the (nonlinear) constraints around the current point. SQP methods are now at a very advanced stage [20], with corresponding implementations being able to warm- or cold-start. In particular, they deal with the problem of infeasible linear constraints (this may happen as the linearization around a point of a set of feasible nonlinear constraints is not always feasible), as well as the feasibility of the starting point with respect to the nonlinear constraints. This case is dealt with by elastic programming [21]. In particular, `snopt` does a good job of finding a constraint feasible point out of any given initial point, even for reasonably large-scale NLPs. By starting a local MINLP solver from a constraint feasible starting point, there are better chances that an integer feasible solution may be found.

9.3 The RECIPE Algorithm

Our main algorithm is a heuristic exploration of the problem solution space by means of an alternating search between the relaxed NLP and the exact MINLP. This is a two-phase global optimization method. Its local phase consists in using the SQP algorithm for solving relaxed (nonconvex) NLPs locally; next, the BB algorithm is used for solving exact (nonconvex) MINLPs to feasibility. The global phase of the algorithm is given by a VNS using two separate neighbourhoods for continuous and general integer variables and for binary variables. The former neighbourhoods have hyper-rectangular shape; the latter are based on a LB constraint involving all binary variables.

We consider a (nonconvex) MINLP P given by formulation (9.1), with its continuous relaxation $\bar{P}$. Let $B = \{i \in Z \mid x_i^L = 0 \wedge x_i^U = 1\}$ be the set of indices of the binary variables, and $\bar{B} = \{1, \ldots, n\} \setminus B$ the set of indices of others, including general integer and continuous variables. Let $Q(\bar{x}, k, k_{\max})$ be its reformulation obtained by adding a local branching constraint

$$\sum_{i \in B} (\bar{x}_i(1 - x_i) + (1 - \bar{x}_i)x_i) \leq \left\lceil k \frac{|B|}{k_{\max}} \right\rceil, \tag{9.3}$$

where $\bar{x}$ is a (binary) feasible solution (e.g. obtained at a previous iteration), $k_{\max} \in \mathbb{N}$ and $k \in \{1, \ldots, k_{\max}\}$. At each VNS iteration (with a certain associated parameter k), we obtain an initial point $\tilde{x}$, where $\tilde{x}_i$ is sampled in a hyperrectangular neighbourhood of radius k for $i \in \bar{B}$ (rounding where necessary for $i \in Z \setminus B$) and $\tilde{x}_i$ is chosen randomly for $i \in B$. We then solve the continuous relaxation $\bar{P}$ locally by means of an SQP method using $\tilde{x}$ as a starting point, and obtain $\bar{x}$ (if $\bar{x}$ is not feasible with respect to the constraints of P, then $\bar{x}$ is re-set to $\tilde{x}$ for possibly having a better choice). We then use a BB method for cMINLPs in order to solve $Q(\bar{x}, k, k_{\max})$, obtaining a solution x'. If x' improves on the incumbent x^*, then x^* is replaced by x' and k is reset to 1. Otherwise (i.e. if x' is worse than x^* or if $Q(\bar{x}, k, k_{\max})$ could not be solved), k is increased in a VNS-like fashion. The algorithm is described formally in Algorithm 1.

9.3.1 Hyperrectangular Neighbourhood Structure

We discuss here the neighbourhood structure for $N_k(x)$ for the RECIPE algorithm.

Consider hyperrectangles $H_k(x)$, centered at $x \in \mathbb{R}^n$ and proportional to the hyperrectangle $x^L \leq x \leq x^U$ given by the original variable bounds, such that $H_{k-1}(x) \subset H_k(x)$ for each $k \leq k_{\max}$. More formally, let $H_k(x^*)$ be the hyperrectangle $y^L \leq x \leq y^U$ where, for all $i \notin Z$,

Algorithm 1: The RECIPE algorithm.

```
INPUT: Neighbourhoods N_k(x) for x ∈ R^n;
       maximum neighbourhood radius k_max;
       number L of local searches in each neighbourhood.
OUTPUT: Best solution found x*.
Set x* = x^L/2 + x^U/2
while (!time-based termination condition) do
  Set k ← 1
  while (k ≤ k_max) do
    for (i = 1 to L) do
      Sample a random point x̃ from N_k(x*).
      Solve P̄ using an SQP algorithm from initial point x̃ obtaining x̄
      if (x̄ is not feasible w.r.t. the constraints of P) then
        x̄ = x̃
      end if
      Solve Q(x̄, k, k_max) using a BB algorithm from initial point x̄ obtaining x'
      if (x' is better than x*) then
        Set x* ← x'
        Set k ← 0
        Exit the FOR loop
      end if
    end for
    Set k ← k + 1.
  end while
end while
```

$$y_i^L = x_i^* - \frac{k}{k_{\max}}(x_i^* - x_i^L)$$
$$y_i^U = x_i^* + \frac{k}{k_{\max}}(x_i^U - x_i^*),$$

for all $i \in Z \setminus B$,

$$y_i^L = \lfloor x_i^* - \frac{k}{k_{\max}}(x_i^* - x_i^L) + 0.5 \rfloor$$
$$y_i^U = \lfloor x_i^* + \frac{k}{k_{\max}}(x_i^U - x_i^*) + 0.5 \rfloor,$$

and for all $i \in B$, $y^L = 0$ and $y^U = 1$.

We let $N_k(x) = H_k(x) \backslash H_{k-1}(x)$. This neighbourhood structure defines a set of hyperrectangular nested shells with respect to continuous and general integer variables. Let τ be the affine map sending the hyperrectangle $x^L \leq x \leq x^U$ into the unit L_∞ ball (i.e., hypercube) $\mathcal{B}$ centered at 0, i.e., $\mathcal{B} = \{x : |x_i| \leq 1 \forall i\}$. Let $r_k = \frac{k}{k_{\max}}$ be the radii of the balls $\mathcal{B}_k$ (centered at 0) such that $\tau(H_k(x)) = \mathcal{B}_k$ for each $k \leq k_{\max}$. In order to sample a random vector $\tilde{x}$ in $\mathcal{B}_k \backslash \mathcal{B}_{k-1}$ we proceed as in Algorithm 2.

The sampled point $\tilde{x}$ will naturally not be feasible in the constraints of (9.1), but we can enforce integral feasibility by rounding $\tilde{x}_j$ to the nearest

integer for $j \in Z$, i.e. by setting $\tilde{x}_j \leftarrow \lfloor \tilde{x}_j + 0.5 \rfloor$. This will be rather ineffective with the binary variables x_j, which would keep the same value $\tilde{x}_j = x_j^*$ for each $k \leq \frac{k_{\max}}{2}$. Binary variables are best dealt with by solving the LB reformulation Q in Algorithm 1.

9.4 Computational Results

Algorithm 1 presents many implementation difficulties: the problem must be reformulated iteratively with the addition of a different LB constraint at each iteration; different solvers acting on different problem formulations must be used. All this must be coordinated by the outermost VNS at the global level. We chose AMPL [19] as a scripting language because it makes it very easy to interface to many external solvers. Since AMPL cannot generate the reformulation Q of P iteratively independently of the problem structure, we employed a C++ program that reads an AMPL output `.nl` file in flat form [29] and outputs the required reformulation as an AMPL-readable `.mod` file.

The `minlp_bb` solver [27] was found to be the MINLP solver that performs best when finding feasible points in nonconvex MINLPs (the comparison was carried out with the default-configured versions of `filMINT` [1] and `BonMin` [6]). The SQP solver of choice was `snopt` [21], found to be somewhat more reliable than `filtersqp` [18]: on the analysed test set, `snopt` achieves, on average, better results at finding feasible solution in a short CPU time. All computational results have been obtained on an Intel Xeon 2.4 GHz with 8 GB RAM running Linux.

RECIPE rests on three configurable parameters: $k_{\max}$ (the maximum neighbourhood radius), L (the number of local searches starting in each neighbourhood) and the maximum allowed user CPU time (not including the time taken to complete the last local search). After some practical experimentation on a reduced subset of instances, we set $k_{\max} = 50$, $L = 15$ and the maximum CPU time to 10h. These parameters were left unchanged over the whole test set, yielding good results without the need for fine-tuning.

Algorithm 2: Sampling in the shell neighbourhoods.

INPUT: $k, k_{\max}$.
OUTPUT: A point $\tilde{x}$ sampled in $H_k(x) \backslash H_{k-1}(x)$.
Sample a random direction vector $d \in \mathbb{R}^n$
Normalize d (i.e., set $d \leftarrow \frac{d}{||d||_\infty}$)
Let $r_{k-1} = \frac{k-1}{k_{\max}}$, $r_k = \frac{k}{k_{\max}}$
Sample a random radius $r \in [r_{k-1}, r_k]$ yielding a uniformly distributed point in the shell
Let $\tilde{x} = \tau^{-1}(rd)$

9.4.1 MINLPLib

The MINLPLib [9] is a collection of Mixed Integer Nonlinear Programming models which can be searched and downloaded for free. Statistics for the instances in the MINLPLib are available at `http://www.gamsworld.org/minlp/minlplib/minlpstat.htm`. The instance library is available at `http://www.gamsworld.org/minlp/minlplib.htm`. The MINLPLib is distributed in GAMS [8] format, so we employed an automatic translator to cast the files in AMPL format.

At the time of downloading (Feb. 2008), the MINLPLib consisted of 265 MINLP instances contributed by the scientific and industrial OR community. These were all tested with the RECIPE algorithm implementation described above. We had 20 unsuccessful runs due to some AMPL-related errors (the model contained some unusual AMPL operator not implemented by some of the solvers/reformulators employed in RECIPE). The instances leading to AMPL-related failure were:

`blendgap`, `dosemin2d`, `dosemin3d`, `fuzzy`, `hda`, `meanvarxsc`, `pb302035`, `pb302055`, `pb302075`, `pb302095`, `pb351535`, `pb351555`, `pb351575`, `pb351595`, `water3`, `waterful2`, `watersbp`, `waters`, `watersym1`, `watersym2`.

The performance of RECIPE was evaluated on the 245 runs that came to completion. The results are given in Tables 9.1, 9.2 (solved instances) and 9.3 (unsolved instances). Table 9.1 lists results where the best solution found by RECIPE was different by at least 0.1% from that listed in MINLPLib. The first column contains the instance name, the second contains the value f^* of the objective function found by the RECIPE algorithm and the third the corresponding CPU usage measured in seconds of user time; the fourth contains the value $\bar{f}$ of the objective function reported in the official MINLPLib table and the fifth contains the name of the corresponding GAMS solver that found the solution. Table 9.2 lists instance names where the best values found by RECIPE and listed in MINLPLib are identical.

9.4.1.1 Optimality

RECIPE found feasible solutions for 163 instances out of 245 (66%). Relative to this reduced instance set, it found the best known solution for 121 instances (74%), gave evidence of the unboundedness of three instances (1%), and improved the best known objective value for 12 instances (7%). In the other cases it found a local optimum that was worse than the best known solution.

Improved solutions were found for the following instances:

Table 9.1 Computational results on MINLPLib. Values denoted by * mark instances with unbounded values in the optimal solution.

instance	RECIPE		known solution	
	f^*	*CPU*	$\bar{f}$	Solver
csched2a	**-165398.701331**	75.957500	-160037.701300	BonMin
eniplac	-131926.917119	113.761000	**-132117.083000**	SBB+CONOPT
ex1233	160448.638212	3.426480	**155010.671300**	SBB+CONOPT
ex1243	118489.866394	5.329190	**83402.506400**	BARON
ex1244	211313.560000	7.548850	**82042.905200**	SBB+CONOPT
ex1265a	15.100000	9.644530	**10.300000**	BARON
ex3	**-53.990210**	1.813720	68.009700	SBB+CONOPT
ex3pb	**-53.990210**	1.790730	68.009700	SBB+CONOPT
fo7_2	22.833307	23.710400	**17.748900**	AlphaECP
fo7	24.311289	25.423100	**20.729700**	AlphaECP
fo9	38.500000	46.296000	**23.426300**	AlphaECP
fuel	17175.000000	1.161820	**8566.119000**	SBB+CONOPT
gear4	1.968201	9.524550	**1.643400**	SBB+CONOPT2
lop97ic	4814.451760	3047.110000	**4284.590500**	-
lop97icx	**4222.273030**	1291.510000	4326.147700	SBB+CONOPT
m7	220.530055	17.275400	**106.756900**	AlphaECP
minlphix	**209.149396***	4.849260	316.692700	SBB+snopt
nuclear14b	**-1.119531**	7479.710000	-1.113500	SBB+CONOPT
nuclear24b	**-1.119531**	7483.530000	-1.113500	SBB+CONOPT
nuclear25	**-1.120175**	1329.530000	-1.118600	SBB+CONOPT
nuclearva	-1.008822	167.102000	**-1.012500**	SBB+CONOPT2+snopt
nuclearvb	-1.028122	155.513000	**-1.030400**	SBB+CONOPT2+snopt
nuclearvc	**-1.000754**	176.075000	-0.998300	SBB+CONOPT2+snopt
nuclearvd	**-1.033279**	202.416000	-1.028500	SBB+CONOPT2+snopt
nuclearve	-1.031364	193.764000	**-1.035100**	SBB+CONOPT2+snopt
nuclearvf	**-1.020808**	200.154000	-1.017700	SBB+CONOPT2+snopt
nvs02	**5.964189**	1.925710	5.984600	SBB+CONOPT3
nvs05	28.433982	4.215360	**5.470900**	SBB+CONOPT3
nvs14	**-40358.114150**	2.070690	-40153.723700	SBB+CONOPT3
nvs22	28.947660	4.849260	**6.058200**	SBB+CONOPT3
o7_2	125.907318	23.262500	**116.945900**	AlphaECP
o7	160.218617	24.267300	**131.649300**	AlphaECP
oil	-0.006926	389.266000	**-0.932500**	SBB+CONOPT(fail)
product	-1971.757941	2952.160000	**-2142.948100**	DICOPT+CONOPT3/CPLEX
st_e13	2.236072	0.548916	**2.000000**	BARON
st_e40	52.970520	0.930858	**30.414200**	BARON
stockcycle	120637.913333	17403.200000	**119948.688300**	SBB+CONOPT
super3t	-0.674621	38185.500000	**-0.685965**	SBB+CONOPT
synheat	186347.748738	3.534460	**154997.334900**	SBB+CONOPT
tln7	19.300000	1000.640000	**15.000000**	BARON
risk2b	$-\infty^*$	45.559100	-55.876100	SBB+CONOPT3
risk2bpb	$-\infty^*$	48.057700	-55.876100	SBB+CONOPT3

csched2a: $f^* = -165398.701331$ (best known solution: -160037.701300)
ex3: $f^* = -53.990210$ (best known solution: 68.009700)
ex3pb: $f^* = -53.990210$ (best known solution: 68.009700)
lop97icx: $f^* = 4222.273030$ (best known solution: 4326.147700)
minlphix: $f^* = 209.149396$ (best known solution: 316.692700)
nuclear14b: $f^* = -1.119531$ (best known solution: -1.113500)
nuclear24b: $f^* = -1.119531$ (best known solution: -1.113500)
nuclear25: $f^* = -1.120175$ (best known solution: -1.118600)
nuclearvc: $f^* = -1.000754$ (best known solution: -0.998300)
nuclearvd: $f^* = -1.033279$ (best known solution: -1.028500)
nuclearvf: $f^* = -1.020808$ (best known solution: -1.017700)
nvs02: $f^* = 5.964189$ (best known solution: 5.984600)
nvs14: $f^* = -40358.114150$ (best known solution: -40153.723700)
risk2b: $f^* = -\infty$ (best known solution: -55.876100)
risk2bpb: $f^* = -\infty$ (best known solution: -55.876100).

Table 9.2 Instances for which RECIPE's optima are the same as those reported in MINLPLib.

alan	ex1224	gbd	nvs06	parallel	st_e32	tln2
batchdes	ex1225	gear2	nvs07	prob02	st_e36	tln4
batch	ex1226	gear3	nvs08	prob03	st_e38	tln5
cecil_13	ex1252a	gear	nvs09	prob10	st_miqp1	tln6
contvar	ex1252	gkocis	nvs10	procsel	st_miqp2	tloss
csched1a	ex1263a	hmittelman	nvs11	pump	st_miqp3	tls2
csched1	ex1263	johnall	nvs12	qap	st_miqp4	util
csched2	ex1264a	m3	nvs13	ravem	st_miqp5	
du-opt5	ex1264	m6	nvs15	ravempb	st_test1	
du-opt	ex1265	meanvarx	nvs16	sep1	st_test2	
enpro48	ex1266a	nuclear14a	nvs17	space25a	st_test3	
enpro48pb	ex1266	nuclear14	nvs18	space25	st_test4	
enpro56	ex4	nuclear24a	nvs19	spectra2	st_test6	
enpro56pb	fac1	nuclear24	nvs20	spring	st_test8	
ex1221	fac2	nuclear25a	nvs21	st_e14	st_testgr1	
ex1222	fac3	nuclear25b	nvs23	st_e15	st_testph4	
ex1223a	feedtray2	nvs01	nvs24	st_e27	synthes1	
ex1223b	feedtray	nvs04	oaer	st_e29	synthes2	
ex1223	gastrans	nvs03	oil2	st_e31	synthes3	

All new best solutions were double-checked for constraint, bounds and integrality feasibility besides the verifications provided by the local solvers, and were all found to be integral feasible; 11 out of 12 were constraint/bound feasible to within a 10^{-5} absolute tolerance, and 1 (`csched2a`) to within 10^{-2}. The 3 instances marked by * in Table 9.1 (`minlphix`, `risk2b`, `risk2bpb`) gave solutions x^* with some of the components at values in excess of 10^{18}. Since `minlphix` minimizes a fractional objective function and there are no upper bounds on several of the problem variables, the optimum is attained when the variables appearing in the denominators tend towards $+\infty$. We solved `risk2b` and `risk2bpb` several times, setting increasing upper bounds to the unbounded variables: this yielded decreasing values of the objective function, suggesting that these instances are really unbounded (hence the $-\infty$ in Table 9.1).

On 82 instances out of 245 listed in Table 9.3, RECIPE failed to find any local optimum within the allotted time limit. Most of these failures are due to the difficulty of the continuous relaxation of the MINLPs: there are several instances where the SQP method (`snopt`) does not manage to find a feasible starting point, and in these cases the convex MINLP solver (`minlp_bb`) also fails. On a smaller number of instances, `minlp_bb` is not able to find integral feasible solutions even though constraint feasible solutions are provided by `snopt`.

9.4.1.2 Reliability

One interesting feature of RECIPE is its reliability: in its default configuration it managed to find solutions with better or equal quality than those

Table 9.3 Instances unsolved by RECIPE.

4stufen	elf	fo9_ar2_1	no7_ar2_1	nuclear49	st_e35	tltr
beuster	fo7_ar2_1	fo9_ar25_1	no7_ar25_1	o7_ar2_1	st_test5	uselinear
deb10	fo7_ar25_1	fo9_ar3_1	no7_ar3_1	o7_ar25_1	st_testgr3	var_con10
deb6	fo7_ar3_1	fo9_ar4_1	no7_ar4_1	o7_ar3_1	super1	var_con5
deb7	fo7_ar4_1	fo9_ar5_1	no7_ar5_1	o7_ar4_1	super2	waste
deb8	fo7_ar5_1	gasnet	nous1	o7_ar5_1	super3	water4
deb9	fo8_ar2_1	m7_ar2_1	nous2	o8_ar4_1	tln12	waterx
detf1	fo8_ar25_1	m7_ar25_1	nuclear104	o9_ar4_1	tls12	waterz
eg_all_s	fo8_ar3_1	m7_ar3_1	nuclear10a	ortez	tls4	windfac
eg_disc2_s	fo8_ar4_1	m7_ar4_1	nuclear10b	qapw	tls5	waterx
eg_disc_s	fo8_ar5_1	m7_ar5_1	nuclear49a	saa_2	tls6	
eg_int_s	fo8	mbtd	nuclear49b	space960	tls7	

reported in the MINLPLib on 136 instances over 245 (55%) and at least a feasible point in a further 11% of the cases. On the same set of test instances, the closest competitor is SBB+CONOPT, which matches or surpasses the best solutions in MINLPLib in 37% of the cases, followed by BARON with 15% and by AlphaECP with 14% (these percentages were compiled by looking at `http://www.gamsworld.org/minlp/minlplib/points.htm` in June 2008).

9.4.1.3 Speed

The total time taken for solving the whole MINLPLib (including the unsolved instances, where the VNS algorithm terminates after exploring the neighbourhoods up to $k_{\max}$ or when reaching the 10 hours time limit, whichever comes first) is roughly 4 days and 19 hours of user CPU time. RECIPE's speed is very competitive with that of sBB approaches: tests conducted using the *ooOPS* solver [28, 29, 34] as well as BARON on some complex MINLPs showed that sBB methods may take a long time to converge. Naturally, the trade-off for this speed is the lack of an optimality guarantee.

9.5 Conclusion

This chapter describes a heuristic approach to solving nonconvex MINLPs based on the mathematical programming formulation. Our approach, called RECIPE, combines several existing exact, approximate and heuristic techniques in a smart way, resulting in a method that can successfully solve many difficult MINLPs without hand-tuned parameter configuration. Such a reliable solver would be particularly useful in industrial applications where the optimum quality is of relative importance and the optimization layer is hidden from user intervention and is therefore "just supposed to work".

Acknowledgements We are very grateful to Prof. Tapio Westerlund for carefully checking all the computational results and informing us of some misprints on the MINLPLib website.

References

1. K. Abhishek, S. Leyffer, and J. Linderoth. FilMINT: An outer-approximation based solver for nonlinear mixed-integer programs. Technical Report ANL/MCS-P1374-0906, Argonne National Laboratory, 2007.
2. C.S. Adjiman, I.P. Androulakis, and C.A. Floudas. Global optimization of MINLP problems in process synthesis and design. *Computers & Chemical Engineering*, 21:S445–S450, 1997.
3. M. Aouchiche, J.M. Bonnefoy, A. Fidahoussen, G. Caporossi, P. Hansen, L. Hiesse, J. Lacheré, and A. Monhait. VNS for extremal graphs 14: The AGX 2 system. In Liberti and Maculan [33], pages 281–308.
4. P. Bonami, L.T. Biegler, A.R. Conn, G. Cornuéjols, I.E. Grossmann, C.D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. Technical Report RC23771, IBM Corporation, 2005.
5. P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A feasibility pump for mixed integer nonlinear programs. Technical Report RC23862 (W0602-029), IBM Corporation, 2006.
6. P. Bonami and J. Lee. `BONMIN` user's manual. Technical report, IBM Corporation, June 2007.
7. J. Brimberg and N. Mladenović. A variable neighbourhood algorithm for solving the continuous location-allocation problem. *Studies in Location Analysis*, 10:1–12, 1996.
8. A. Brook, D. Kendrick, and A. Meeraus. GAMS, a user's guide. *ACM SIGNUM Newsletter*, 23(3-4):10–11, 1988.
9. M.R. Bussieck, A.S. Drud, and A. Meeraus. MINLPLib — a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1), 2003.
10. ARKI Consulting and Development. *SBB Release Notes*, 2002.
11. M. Dražic, V. Kovačević-Vujčić, M. Čangalović, and N. Mladenović. Glob — a new VNS-based software for global optimization. In Liberti and Maculan [33], pages 135–154.
12. M. Dražić, C. Lavor, N. Maculan, and N. Mladenović. A continuous variable neighbourhood search heuristic for finding the tridimensional structure of a molecule. *European Journal of Operational Research*, 185:1265–1273, 2008.
13. A. Drud. CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems. *Mathematical Programming*, 31:153–191, 1985.
14. M. Duran and I. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.
15. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–37, 2005.
16. R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.
17. R. Fletcher and S. Leyffer. Numerical experience with lower bounds for MIQP branch-and-bound. *SIAM Journal of Optimization*, 8(2):604–616, 1998.
18. R. Fletcher and S. Leyffer. User manual for `filter`. Technical report, University of Dundee, UK, March 1999.
19. R. Fourer and D. Gay. *The AMPL Book*. Duxbury Press, Pacific Grove, 2002.
20. P. Gill, W. Murray, and M.A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal of Optimization*, 12(4):979–1006, 2002.

21. P.E. Gill. *User's guide for SNOPT version 7*. Systems Optimization Laboratory, Stanford University, California, 2006.
22. P. Hansen and N. Mladenović. Variable neighbourhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.
23. P. Hansen and N. Mladenović. Variable neighbourhood search. In P. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*. Oxford University Press, Oxford, 2002.
24. P. Hansen and N. Mladenović. Variable neighbourhood search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer, Dordrecht, 2003.
25. P. Hansen, N. Mladenović, and D. Urošević. Variable neighbourhood search and local branching. *Computers & Operations Research*, 33(10):3034–3045, 2006.
26. C. Lavor, L. Liberti, and N. Maculan. Computational experience with the molecular distance geometry problem. In J. Pintér, editor, *Global Optimization: Scientific and Engineering Case Studies*, pages 213–225. Springer, Berlin, 2006.
27. S. Leyffer. User manual for `minlp_bb`. Technical report, University of Dundee, UK, March 1999.
28. L. Liberti. *Reformulation and Convex Relaxation Techniques for Global Optimization*. PhD thesis, Imperial College London, UK, March 2004.
29. L. Liberti. Writing global optimization software. In Liberti and Maculan [33], pages 211–262.
30. L. Liberti and M. Dražic. Variable neighbourhood search for the global optimization of constrained NLPs. In *Proceedings of GO Workshop, Almeria, Spain*, 2005.
31. L. Liberti, C. Lavor, and N. Maculan. Double VNS for the molecular distance geometry problem. In P. Hansen, N. Mladenović, J.A. Moreno Pérez, editors, *Proceeding of the 18th Mini Euro Conference on Variable Neighbourhood Search*, Tenerife, Spain, 2005.
32. L. Liberti, C. Lavor, N. Maculan, and F. Marinelli. Double variable neighbourhood search with smoothing for the molecular distance geometry problem. *Journal of Global Optimization*, accepted for publication.
33. L. Liberti and N. Maculan, editors. *Global Optimization: from Theory to Implementation*. Springer, Berlin, 2006.
34. L. Liberti, P. Tsiakis, B. Keeping, and C.C. Pantelides. *ooOPS*. Centre for Process Systems Engineering, Chemical Engineering Department, Imperial College, London, UK, 2001.
35. A. Lodi. Personal communication, 2007.
36. N. Mladenović, J. Petrović, V. Kovačević-Vujčić, and M. Čangalović. Solving a spread-spectrum radar polyphase code design problem by tabu search and variable neighbourhood search. *European Journal of Operational Research*, 151:389–399, 2003.
37. J. Puchinger and G.R. Raidl. Relaxation guided variable neighbourhood search. In *Proc. of Mini Euro Conference on Variable Neighbourhood Search, Tenerife, Spain*, 2005.
38. E.M.B. Smith and C.C. Pantelides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. *Computers & Chemical Engineering*, 23:457–478, 1999.
39. M. Tawarmalani and N.V. Sahinidis. Global optimization of mixed integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004.
40. T. Westerlund. Some transformation techniques in global optimization. In Liberti and Maculan [33], pages 45–74.
41. T. Westerlund and R. Pörn. Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. *Optimization and Engineering*, 3:235–280, 2002.
42. T. Westerlund, H. Skrifvars, I. Harjunkoski, and R. Pörn. An extended cutting plane method for a class of non-convex MINLP problems. *Computers & Chemical Engineering*, 22(3):357–365, 1998.

Chapter 10
Variable Intensity Local Search

Snežana Mitrović-Minić and Abraham P. Punnen

Abstract This chapter considers a local search based heuristic framework for solving the mixed-integer programming problem (MIP) where a general purpose MIP solver is employed to search the associated neighborhoods. The associated neighborhood search problems are MIPs of smaller sizes. The neighborhoods are explored in varying the intensity by changing time and size parameters. This local search can be viewed as a combination of very large scale neighborhood (VLSN) search and variable neighborhood search (VNS). The approach has been implemented to solve two integer programming problems: the generalized assignment problem, and the multi-resource generalized assignment problem. Encouraging computational results have been achieved.

10.1 Introduction

In this chapter we consider a local search algorithm for the mixed-integer programming problem (MIP) based on the well known k-exchange neighborhood. Unlike traditional k-exchange based local search that considers small values of k, we use large values k. An MIP solver is used to explore the neighborhoods for improved solutions. The neighborhoods size and search intensity are controlled by two search-intensity parameters. Our algorithm in many cases does not explore the k-exchange neighborhood optimally but performs only an approximate search. Thus, we are exploring only a partial k-exchange neighborhood, for various values of k.

Several local search algorithms from the optimization literature that partially explore k-exchange neighborhoods—and are classified as variable depth methods—include the Lin-Kernighan algorithm for TSP [11] and ejection

Snežana Mitrović-Minić · Abraham P. Punnen
Department of Mathematics, Simon Fraser University, Surrey, Canada
e-mail: {snezanam,apunnen}@sfu.ca

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_10,

chain algorithms for various combinatorial optimization problems [10]. Although our algorithm is a local search, considering its linkages with VLSN search [1, 2] and VNS [16], we call it *variable intensity local search* (VILS).

Using an MIP solver within local search to explore neighborhoods received considerable attention in the recent years: [5, 6] for the general MIP, [3, 22, 21, 23] for variations of the vehicle routing problems, [18, 15, 14] for the variations of the generalized assignment problem. The algorithm discussed in this chapter is a generalization of the heuristics developed for the generalized assignment problem (GAP) [15] and the multi-resource generalized assignment problem (MRGAP) [14].

This chapter is organized as follows. In Section 10.2 we introduce the general VILS framework. Section 10.3 gives brief description of our experimental studies on GAP and MRGAP problems whose details are reported in [15, 14]. Concluding remarks are given in Section 10.4.

10.2 The General VILS Framework

The VILS algorithm is a local search algorithm for MIP using the k-exchange neighborhood for varying values of k, adjusted systematically during the algorithm. The resulting neighborhoods are searched approximately using an MIP-solver with varying intensity level. Consider the MIP

$$\begin{array}{lll} \text{MIP:} & \text{Maximize} & CX \\ & \text{Subject to} & AX = b \\ & & X \geq 0, \quad X \text{ integer,} \end{array}$$

where $X^T = (x_1, x_2, \ldots, x_n)$ is a vector of n variables, and the problem parameters are: $A = (a_{ij})$ which is an $m \times n$ matrix, $b^T = (b_1, b_2, \ldots, b_m)$ which is an m-vector, and $C = (c_1, c_2, \ldots, c_n)$ which is an n-vector. For simplicity of presentation, we avoid real variables (which are never set to a fixed value during the course of the algorithm) in the above description of an MIP. Let $\hat{X}$ be a feasible solution to the MIP. A *binding set* $\hat{S}$ is a subset of variable indices $\{1, 2, \ldots, n\}$ which defines a k-exchange neighborhood. The neighborhood $N(\hat{X})$ consists of all solutions of the MIP whose j^{th} variable is equal to the value of the j^{th} variable in $\hat{X}$ for all $j \in \hat{S}$, *i.e.*

$$N(\hat{X}) = \{X \mid x_j = \hat{x}_j, \ \forall j \in \hat{S} \text{ and } X \text{ is a feasible solution to the MIP}\}$$

The neighborhood $N(\hat{X})$ can be searched for an improving solution by solving the following restricted MIP

$$\begin{array}{lll} \text{MIP}(\hat{S}): & \text{Maximize} & \displaystyle\sum_{j \in N \setminus \hat{S}} c_j x_j \\ & \text{Subject to} & \end{array}$$

$$\sum_{j \in N \setminus \hat{S}} a_{ij} x_{ij} = \hat{b}_i, \text{ for } i = 1, 2, \ldots m$$
$$x_j \geq 0, \quad x_j \text{ integer for } j \in N \setminus \hat{S}$$

where $\hat{b}_i = b_i - \sum_{j \in \hat{S}} a_{ij} \hat{x}_{ij}$. If $\bar{X} = \{\bar{x}_j \ : \ j \in N \setminus \hat{S}\}$ is a feasible solution to MIP($\hat{S}$) then the n-vector X defined by

$$x_j = \begin{cases} \hat{x}_j, & \text{if } j \in \hat{S} \\ \bar{x}_j, & \text{otherwise} \end{cases}$$

is a feasible solution to MIP and $X \in N(\hat{X})$. We call such a solution X *the solution augmented by* $\bar{X}$.

The complexity of MIP($\hat{S}$) in practice depends primarily on the size of $\hat{S}$ although other factors are also involved. If $|\hat{S}|$ is large (and hence $|N \setminus \hat{S}|$ is small) MIP($\hat{S}$) can normally be solved optimally in reasonable time using an MIP solver. However, in this case, $|N(\hat{X})|$ is likely to be small, limiting the power of the local search. If $|\hat{S}|$ is small, then $|N(\hat{X})|$ is likely to be large yielding a more powerful search but the time for searching the neighborhood using the MIP solver could be large. Thus the efficiency of the local search using the MIP solver depends on our ability to guide the search appropriately by controlling the size of $\hat{S}$, the time invested in searching $N(\hat{X})$, and the choice of elements in $\hat{S}$.

In the VILS algorithm, we keep six major parameters: p is the cardinality of the binding set $\hat{S}$, p_0 is its initial value, Δp is the downward step size to decrease the value of p; t is the time limit for the MIP solver, t_0 is its initial value, and Δt is the upward step size of t.

Initially, we set a large value of p yielding smaller neighborhoods. The search times for these neighborhoods are set to small values and the local search is carried out until a decision is made to intensify the search. At this stage, the value of p is decreased (and thereby increasing the neighborhood size) and the time limit for searching the neighborhood is increased. This process is continued until a prescribed stopping criterion is reached.

The control mechanism using the time limits and the systematic intensification of the search resulted in good experimental results. Different selections of the binding sets $\hat{S}$ yield different neighborhoods, and they are normally problem specific. Assume that L is the number of different neighborhoods. If no improvement is obtained after employing several binding set selection rules, the search intensity is increased by decreasing p and increasing t. A high level description of the VILS algorithm is given in Figure 10.1.

The neighborhoods and intensity-search schemata are problem specific. Neighborhoods may be designed using any existing or new strategy for choosing a binding set. Our strategies for choosing binding sets may be summarized as follows. A criterion for "good" variables is chosen beforehand, and variables are fixed in order of "goodness". In the iteration where p variables

```
The VILS Algorithm
Input: Problem instance P;
    the stopping criterion and the intensity-search change criterion;
    p_0, Δp; t_0, Δt
begin
    generate feasible solution X̂
    i ⇐ 0
    p ⇐ p_0
    t ⇐ t_0
    while (the stopping criterion is not satisfied) do
        choose the binding set S_i such that |S_i| = p and
        generate the neighborhood N_i
        /* search the neighborhood */
        Solve the problem MIP(S_i) by running the MIP solver for time t
        Let X̄ be the best solution obtained
        Compute the augmented solution X'
        /* update the current solutions */
        if (CX' < CX̂) then
            X̂ ⇐ X'
        end if
        i ⇐ (i + 1) mod L
        if (the intensity-search change criterion is satisfied) then
            p ⇐ p − Δp
            t ⇐ t + Δt
        end if
    end while
    return X̂
end
```

Fig. 10.1 Outline of the VILS Algorithm.

have to be fixed, the following are the neighborhoods for the GAP and the MRGAP with m machines.

1. For each machine, fix p/m "best" variables out of the value-one variables.
2. For each machine, fix p/m "worst" of the value-one variables.
3. For each machine, fix $p/(m/2)$ "best" and $p/(m/2)$ "worst" of the value-one variables.
4. For half of the machines, fix p/m "best", and for the other half of the machines, fix p/m "worst" of the value-one variables.
5. Fix "best" p of the value-one variables.
6. Fix "worst" p of the value-one variables.
7. Fix $p/2$ "best" and $p/2$ "worst" of the value-one variables.
8. Controlled random fixing: fix $p/10$ random variables in the "best" 10% of the value-one variables, fix $p/10$ random variables in the next "best" 10% (11% to 20%) of the value-one variables, etc.
9. Meta-neighborhood: fix certain sequences of the value-one variables in the given "goodness" order.

The variable "goodness" criteria depends only on the initial problem parameters, and thus the variables can be ordered by their goodness in a preprocessing step. An example of a "goodness" criterion we used for the GAP is: A "good" variable is one with smaller ratio cost per resource needed. Further details about the neighborhoods used in our two experimental studies may be found in [14, 15].

Since the truncation of the current solution $\hat{X}$ is a feasible solution to MIP(S_i), we supply it to the MIP solver. To test the efficiency of the algorithm we considered two specific problems: the GAP which is well studied in literature [4, 12, 13, 19, 24, 25] and its generalization the MR-GAP [7, 8, 9, 17, 20]. Our experimental studies, algorithm parameters, and results are summarized in the next section.

10.3 Experimental Studies

We have implemented the VILS for the GAP and MRGAP in C++ and tested on a Dell workstation with one Intel Xeon 2.0GHz processor, 512 MB memory, GNU g++ compiler version 3.2, and Linux (Mandrake 9.2) operating system. To search the neighborhoods we have used CPLEX 9.1 with Concert Technology.

The stopping criterion has been taken according to the time limits used in [24, 26, 25]. The intensity-search change criterion has been: "solution has not improved in 3 iterations" although we also experimented with values 2, 4 and 5. Preliminary studies have also shown that an appropriate number of different neighborhoods (the binding strategies) L should be between 4 and 10, when the intensity-search change criterion is 2 or 3 to assure that each neighborhood type is searched once in every two or three intensity settings.

We have experimented with different intensity schedules with variety of combinations of changing time limits and binding set size alternatively and simultaneously. However, more complicated schedules, as well as more granular schemas, have not shown any additional advantages. When a number of iterations does not generate an improving solution, the simple increase in time limit and neighborhood size almost always produces improved solution. Further research towards reactive VILS is in progress, where initial neighborhood size and time limit as well as the steps would be chosen automatically.

For the GAP, standard benchmark large instances of types C, D, and E, with 900 and 1600 jobs, generated by [24] were used as the test bed. Nine out of eighteen solutions achieved by VILS were equal or better in quality compared to the solutions when tabu search by [25] was run only once. (Six solutions were better.) When tabu search was run five times [25], it achieved better results for all but two instance. In other five instances the solutions were the same.

For the MRGAP, our testbed consists of MRGAP instances generated by [26] from the standard benchmark GAP instances of types C, D and E with 100 and 200 tasks (which were generated by J.E. Beasley). We have tested the VILS with two different intensification schemes: $Sch1$ and $Sch2$ (details may be found in [15, 14]). The solutions achieved by VILS are better or equal in quality compared to the solutions reported in the literature, with a few exceptions when tabu search by [26] or CPLEX achieved better solutions.

For the D instances, the best solutions were achieved by VILS with intensification schedule $Sch1$ in 7 cases, by VILS with intensification schedule $Sch2$ in 10 cases, and by CPLEX in 10 cases. Unique best solutions were achieved by VILS ($Sch1$), VILS ($Sch1$), and CPLEX in 6, 8, and 7 instances, respectively. For the E instances, the best solutions were achieved by VILS ($Sch1$), VILS ($Sch1$), tabu search [26], and CPLEX in 16, 14, 8, and 12 instances, respectively. Unique best solutions were achieved by VILS ($Sch1$), VILS ($Sch1$), tabu search [26], and CPLEX in 6, 3, 2, and 1 instances, respectively.

10.4 Conclusion

In this chapter we proposed an implementation of a local search framework, called Variable Intensity Local Search, for solving mixed-integer programming problems. The neighborhoods are explored using a general purpose MIP solver. Depending on the binding sets, the neighborhoods could be of different structure and hence the algorithm can be viewed as a variable neighborhood search. In addition, since some of the search neighborhoods could be very large, the algorithm can be viewed as a very large scale neighborhood search as well. We have done two experimental studies solving GAP and MRGAP which showed that good quality solutions can be reached in a reasonable time. We are in the process of conducting an experimental study on a facility location problem and on a general 0-1 MIP.

The major advantage of the approach is its local search framework simplicity, and ability to achieve satisfactory results by controlling the intensity and depth of the neighborhood search. Furthermore, our heuristic can be embedded in any metaheuristic.

Acknowledgements This work is partially supported by an NSERC discovery grant awarded to Abraham P. Punnen.

References

1. R.K. Ahuja, O. Ergun, and A. Punnen. A survey of very large scale neighborhood search techniques. *Discrete Applied Mathematics*, 23:75–102, 2002.
2. R.K. Ahuja, O. Ergun, and A. Punnen. Very large scale neighborhood search: Theory, algorithms, and applications. In T. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, volume 10 of *Computer and Information Science Series*. Chapmann and Hall, CRC Press, 2007.
3. R. Bent and P. V. Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33:875–893, 2006.
4. D. Cattrysse and L.N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60:260–272, 1992.
5. E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102:71–90, 2005.
6. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.
7. B. Gavish and H. Pirkul. Allocation of databases and processors in a distributed computing system. In J. Akoka, editor, *Management of Distributed Data Processing*. North-Holland Publishing Company, Amsterdam, 1982.
8. B. Gavish and H. Pirkul. Computer and database location in distributed computer systems. *IEEE Transactions in Computing*, 35:583–590, 1986.
9. B. Gavish and H. Pirkul. Algorithms for the multi-resource generalized assignment problem. *Management Science*, 37:695–713, 1991.
10. F. Glover. New ejection chain and alternating path methods for traveling salesman problem. In O. Balci, R. Sharda, and S. Zenios, editors, *Computer Science and Operations Research: New Development in Their Interfaces*, pages 491–507. Pergamon, Oxford, 1992.
11. S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.
12. H.R. Lourenço and D. Serra. Adaptive approach heuristic for the generalized assignment problem. Technical report, Department of Economics and Management, Universitat Pompeu Fabra, R. Trias Fargas 25-27, 08005 Barcelona, Spain, 1998.
13. S. Martello and P. Toth. An algorithm for the generalized assignment problem. In J.P. Brans, editor, *Operational Research'81*, pages 589–603. North-Holland, 1981.
14. S. Mitrovic-Minic and A.P. Punnen. Local search intensified: Very large-scale variable neighborhood search for the multi-resource generalized assignment problem. Submitted for publication.
15. S. Mitrovic-Minic and A.P. Punnen. Very large-scale variable neighborhood search for the generalized assignment problem. *Journal of Interdisciplinary Mathematics*, 11(5):653–670, 2008.
16. N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.
17. R.A. Murphy. A private fleet model with multi-stop backhaul. Working paper 103, Optimal Decision Systems, Green Bay, WI54306, 1986.
18. T. Oncan, S.N. Kabadi, K.P.N. Nair, and A.P. Punnen. VLSN search algorithms for partitioning problems using matching neighbourhoods. *The Journal of the Operational Research Society*, 59:388–398, 2008.
19. I.H. Osman. Heuristics for the generalized assignment problem: simulated annealing and tabu search approaches. *OR Spektrum*, 17:211–225, 1995.
20. H. Pirkul. An integer programming model for allocation of databases in a distributed computer system. *European Journal of Operational Research*, 26:401–411, 1986.
21. D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34:2403–2435, 2007.

22. S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.
23. G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.
24. M. Yagiura, T. Ibaraki, and F. Glover. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing*, 16:133–151, 2004.
25. M. Yagiura, T. Ibaraki, and F. Glover. A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*, 169:548–569, 2006.
26. M. Yagiura, S. Iwasaki, T. Ibaraki, and F. Glover. A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. *Discrete Optimization*, 1(1):87–98, 2004.

Chapter 11
A Hybrid Tabu Search for the m-Peripatetic Vehicle Routing Problem

Sandra Ulrich Ngueveu, Christian Prins, and Roberto Wolfler Calvo

Abstract This chapter presents a hybridization of a perfect b-matching within a tabu search framework for the m-Peripatetic Vehicle Routing Problem (m-PVRP). The m-PVRP models, for example, money transports and cash machines supply where, for security reasons, no path can be used more than once during m periods and the amount of money allowed per vehicle is limited. It consists in finding a set of routes of minimum total cost over m periods from an undirected graph such that each customer is visited exactly once per period and each edge can be used at most once during the m periods. Each route starts and finishes at the depot with a total demand not greater than the vehicle capacity. The aim is to minimize the total cost of the routes. The m-PVRP can be considered as a generalization of two well-known NP-hard problems: the vehicle routing problem (VRP or 1-PVRP) and the m-Peripatetic Salesman Problem (m-PSP). Computational results on classical VRP instances and TSPLIP instances show that the hybrid algorithm obtained improves the tabu search, not only on the m-PVRP in general, but also on the VRP and the m-PSP.

11.1 Introduction

The m-Peripatetic Vehicle Routing Problem (m-PVRP), introduced for the first time in [12], models money collection, transfer and dispatch when it is subcontracted by banks and businesses to specialized companies. These companies need optimized software or applications to organize their van or truck routes and schedule. For security reasons, peripatetic and capacity constraints

Sandra Ulrich Ngueveu · Christian Prins · Roberto Wolfler Calvo
Institut Charles Delaunay – LOSI, Universitè de Technologie de Troyes (UTT), Troyes, France
e-mail: {ngueveus,christian.prins,roberto.wolfler_calvo}@utt.fr

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_11,

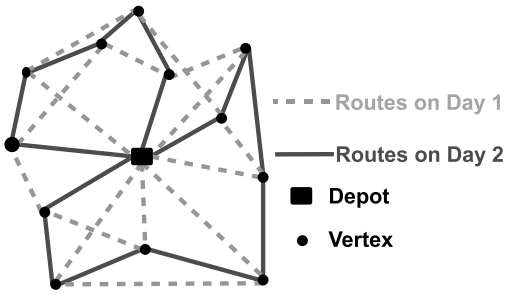


Fig. 11.1 Example of a solution for a 2-PVRP.

must be satisfied: no path can be used more than once during m periods and the amount of money allowed per vehicle is limited. The m-PVRP is defined on a complete graph $G = (V, E)$ where V is the vertex set and E is the edge set. It consists in finding a set of routes of minimum total cost over m periods from an undirected graph such that each customer is visited exactly once per period and each edge can be used at most once during the m periods. Figure 11.1 shows an example of a feasible solution for a 2-PVRP. Ngueveu et al. introduced the m-PVRP before proposing two lower bounds and two upper bounds. The two lower bounds are based upon k edge-disjoint spanning trees and a perfect b-matching. The first upper bound results from the adaptation of the Clarke-Wright heuristic [4] and the second from a tabu search with diversification.

The m-PVRP can be considered as a generalization of two well-known NP-hard problems: the vehicle routing problem (VRP) and the m-peripatetic salesman problem (m-PSP). Indeed, the VRP is a particular case of m-PVRP where $m = 1$ since it consists in finding the best routes for one single period. Likewise, any m-PSP is in fact an m-PVRP with an infinite vehicle capacity since the traveling salesman problem (TSP) is a particular case of the VRP with one single vehicle. Both problems were widely studied in the literature with heuristics, metaheuristics and exact methods. The m-PSP, e.g., was introduced by Krarup [11] and mainly studied in [6, 8, 16]. Amongst the numerous publications concerning the VRP, we can cite Toth and Vigo [14], a recent survey of the most effective metaheuristics for VRPs [5], or an effective exact algorithm based on q-route relaxation [2].

In this chapter we present an efficient algorithm resulting from the hybridization of the perfect b-matching and the tabu search of Ngueveu et al. It is designed to solve the m-PVRP. However, due to the lack of publicly available instances for this new problem, the computational analysis was

performed using instances of the VRP and the m-PSP to compare with the literature. The remainder of this paper is organized as follows. Section 11.2 presents the tabu components, while Section 11.3 focuses on the hybridization with a b-matching. Finally, the computational evaluation is presented in Section 11.4, before the conclusion.

11.2 Tabu Search

Tabu search [10] is a method that explores the solution space by moving from a solution s_t identified at iteration t to the best solution s_{t+1} in the neighborhood $N(s_t)$. Since s_{t+1} may not improve s_t, a tabu mechanism is implemented to prevent the process from cycling over a sequence of solutions. An obvious way to prevent cycles would be to forbid the process from going back to previously encountered solutions, but doing so would typically require excessive bookkeeping. Instead, some attributes of past solutions are recorded and solutions possessing these attributes are discarded for τ iterations. This mechanism is often referred to as short-term memory. Other features like granularity and diversification (long term memory) are often implemented to improve speed and efficiency. The algorithm we designed is stopped after a predefined number of iterations $maxt$ and requires the following components, described hereafter: the initial solution heuristic, the neighborhood structure, the penalization component and the tabu list management.

11.2.1 Initial Solution Heuristic and Neighborhood Structure

Inspired by the idea of Krarup for the m-PSP [11], the procedure of Clarke and Wright [4] is applied m times to obtain at the end an initial m-PVRP solution, and the edges already used are removed from the graph before each iteration. In practice, a penalty is added to the cost of edges already used, forbidding the reuse of any of them, unless there is no other alternative. This procedure will be referred to as *Heuristic*.

To explore the solution space, we try to introduce into the current solution edges that are not currently used during the m periods. Figure 11.2 illustrates the eight different ways, derived from classical 2-opt moves, to introduce an edge [A, B] within a period. There are consequently $8m$ potential insertion moves per edge. Moves involving two routes are authorized only if the capacity constraints are not violated: the total demand on each of the new routes obtained must not exceed the vehicle capacity Q. In addition to the classical 2-opt neighborhood, this neighborhood authorizes moves that split a route in

two (see cases 3 and 4 on Figure 11.2) or merge two routes if an edge inserted connects the extremities of two routes.

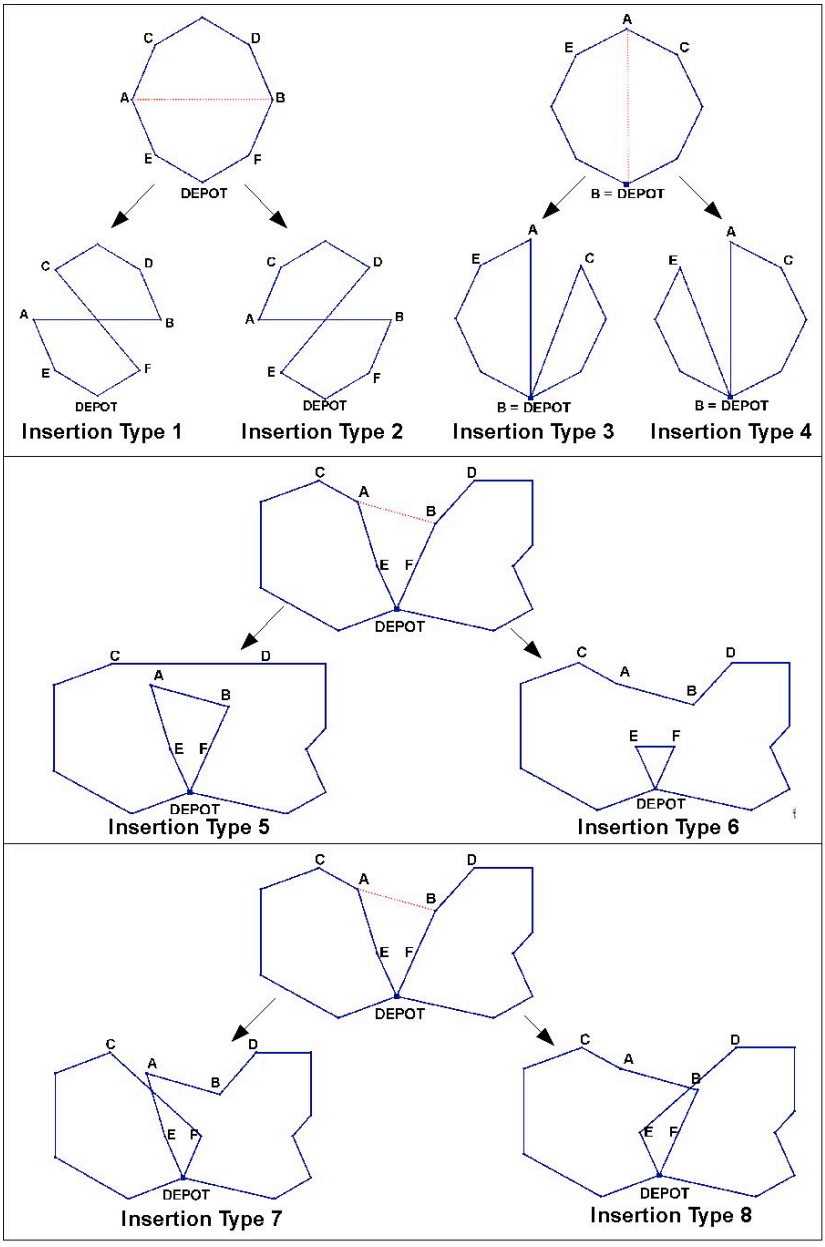


Fig. 11.2 Neighborhood definition: eight ways to insert edge [A,B] during a period.

11.2.2 Penalization and Tabu List Management

To allow our algorithm to start from a non-feasible solution, peripatetic constraints are removed and the penalty $\alpha \times max(0, (\sum_{k\in\mathbb{K}} x_e - 1))$ is added to the objective function. Consequently, an edge may be used more than once during two or more different periods. Within the hybrid tabu search, we set α to $2\bar{c}_{max}$ where $\bar{c}_{max}$ is the cost of the most expensive edge of the graph.

To avoid going back to already visited solutions, after each iteration t, the edges removed from the solution are inserted in the tabu list TL and are declared tabu until iteration $t + \tau$, where τ is the tabu tenure. During each iteration, an unused and non-tabu edge e has to be inserted with the best possible move and the second entering edge e' is free: e' can be tabu or be already used in a period of the solution, in which case it will be penalized as explained above. The "partial tabu" algorithm obtained in this way is not very sensitive to the value of τ while it avoids cycling. We also applied an aspiration criterion, which consists in authorizing a tabu move when the solution obtained is the best found so far.

11.3 Hybridization with *b*-Matching and Diversification

Hybridization can in our context consist either in using information provided by an exact method to guide the metaheuristic, or in combining the features of two metaheuristics to obtain a more efficient procedure. The hybridization of b-matching with tabu search, as explained in Section 11.3.2, and the diversification procedure, detailed in Section 11.3.3, both improved the speed and efficiency of the tabu search designed for the m-PVRP.

11.3.1 b-Matching

The b-matching problem, also known as the b-directed flow problem, was introduced by Edmonds [9] within the class of well-solved integer linear problems. Define c_e as the cost of edge e, y_e as the binary variable equal to 1 only if edge e is used, and 0 otherwise. If d_i is the demand of node i and Q is the vehicle capacity, then the minimal number of vehicles per period is $\lambda = \left\lceil \frac{1}{Q} \sum_{i\in V} d_i \right\rceil$. The mathematical formulation of the b-matching obtained after relaxing the capacity constraints of the m-PVRP is as follows:

$$\min \sum_{e\in E} c_e y_e$$

s. t.

$$\sum_{e \in \delta(i)} y_e = b_i \quad \text{with } b_i = \begin{cases} 2m & \forall i \in \{1...n\} \\ 2m\lambda \text{ if} & i = 0 \end{cases}$$

$$y_e \in \{0,1\}, \qquad \forall e \in E$$

A solution to this problem can be easily computed with a linear programming solver. Preliminary results from [12] suggested that the value obtained may be on average about 10% less than the optimal m-PVRP solution. Therefore, repairing b-matching solutions could lead to potentially good upper bounds. However, extracting an m-PVRP solution from a set of edges is not a straightforward process because it requires to partition the edges between the m periods and the routes. To overcome this difficulty, we hybridize the b-matching with a tabu search algorithm: the result of the exact method guides the metaheuristic in the solution space.

11.3.2 Hybridization

Granularity is a concept introduced in [15], based on the idea of using restricted neighborhoods. It allows only moves that, based on some criterion, are more likely to produce good feasible solutions. Its implementation for the VRP consists in delaying the introduction of long edges into the solution. In our case, the result of the b-matching is used to define the tabu granularity and guides the metaheuristic in the solution space. The resulting algorithm is a granular tabu search that uses as candidate list the unused edges that are in the b-matching solution; these edges have a higher probability of being part of an optimal solution.

Solving the b-matching produces a set of potentially good edges for the m-PVRP: the cheapest set of edges that satisfy the aggregated degree constraints. However, a small number of edges tends to be selected (e.g. 10% for instance B-n45-k7 for the 2-PVRP). This leads to a very small candidate list, which induces a small neighborhood size, counter-effective for the metaheuristic efficiency. We found two ways to enlarge this neighborhood without losing the advantage of the b-matching data:

1. Relax the integrality constraints of the b-matching: this increases the numbers of edges selected by edges that still have a higher probability than others to be in an optimal solution.
2. Complete the b-matching granularity with a short-edge subset: following Toth and Vigo's primary idea, short edges disregarded by the b-matching are added to the candidate list of edges to be inserted into the current solution.

This latter subset is composed of edges that have a cost not greater than $\mu\bar{c}$ and are currently unused; $\bar{c}$ is the average cost of edges used within the initial

solution and μ is a parameter. The penalty applied to infeasible solutions (see Section 11.2.2) has been included in the computation of $\bar{c}$. The idea behind keeping the penalty in the calculation is that if α was set to 0, the initial infeasible solution may be cheaper than feasible solutions. Therefore, edges included in the candidate list need to be a little more expensive to allow the metaheuristic to find feasible solutions.

The granularity (relaxed b-matching plus short-edge subset) is applied every time the best solution is improved, and removed after $GTSmaxk$ iterations without improving the best solution. During the search, the algorithm oscillates between intensification phases (when granularity is activated: $g = true$) and pseudo-diversification phases (when granularity is removed: $g = false$).

11.3.3 Diversification Procedure

Diversification ensures that the search process is not restricted to a limited portion of the search space. An example of implementation, as explained in [13], penalizes edge costs depending on their frequency of use during the search. For the m-PVRP, we do not want to penalize edges used very often because they might be required to reach an optimal solution. Instead, our diversification procedure searches for the best way to insert into the current solution the cheapest edge unused so far. To accommodate this component with the b-matching granularity, the procedure is applied as soon as the following two conditions are satisfied:

1. At least Max_γ iterations have been performed without improving the best solution since the last removal of the b-matching granularity (described in the previous subsection).
2. The previous move applied was not an improving move.

The diversification component applied in this way does not disturb the b-matching granularity, but gives a helpful "kick" when necessary. Let $f(S)$ be the total cost of solution S, algorithm 1 summarizes the hybrid tabu search with diversification designed for the m-PVRP.

11.4 Computational Analysis

A computational evaluation was first performed on classical VRP and m-PSP benchmark problems to compare our results with the literature; next we applied our algorithms to the m-PVRP with $m > 1$. The tests were done on four classes of VRP instances from the literature: A, B, P and vrpnc. Classes A, B and P [1] contain 27, 23 and 23 instances, respectively, of 19 to 101 nodes. From class vrpnc [3] we selected the seven instances with 50-199 nodes and

Algorithm 1: Hybrid Tabu Search

```
1: Heuristic(S)
2: S' := S
3: TL := ∅; g := true; t := 1; k := 1; Dec := 1; Freq[e] := 0 ∀e ∈ E
4: repeat
5:    FindBestNonTabuSolution(S', TL, g, f(S), Dec)
6:    if f(S') < f(S) then
7:       S := S'
8:       k := 1; γ := 1
9:       if g = false then
10:         g := true
11:      end if
12:   else
13:      γ := γ + 1
14:      if g = true then
15:         k := k + 1
16:         if k > GTSmaxk then
17:            g := false
18:            k := 1; γ := 1
19:         end if
20:      end if
21:      if γ > Max_γ and Dec = −1 then
22:         Diversify(S', Freq)
23:      end if
24:   end if
25:   UpdateTabuList(TL, τ)
26: until t > maxt
```

no additional route length restriction. All VRP instances can be found on the website `http://neo.lcc.uma.es/radi-aeb/WebVRP`. We also used the five Euclidian instances from TSPLIB (`http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`) with 17 to 29 nodes that were already used for the m-PSP in [7].

The experiments were performed on an Intel Core 2 Duo personal computer at 1.80 GHz with 2 GB of RAM running Windows Vista. Metaheuristics were coded in C, but the linear b-matching solution required for granularity was obtained with the open source software GLPK. The tables of this section compare four variants of our algorithms, the basic tabu search algorithm (TS), the tabu search algorithm with the diversification component (TS+D), the tabu search algorithm hybridized with b-matching (HTS) and the latter further enhanced by the diversification component (HTS+D). These algorithms are tested on the VRP, the m-PSP and the m-PVRP with $m > 1$.

Some preliminary experiments were made to tune the parameters of the upper bounding procedures. Preliminary results led to a different *HTS* setting per problem and per class of instances. To limit the number of settings used, we decided to apply the following *HTS* settings of the parameters for each problem solved:

Table 11.1 Parameter Settings

Algo	Param	Description	Value
(H)TS(+D)	α	Penalization	$2\bar{c}_{max}$
(H)TS(+D)	$maxt$	Max number of iterations	10000
(H)TS(+D)	τ	Tabu duration	n
HTS(+D)	μ	Proportion of average edge cost	1.30
HTS(+D)	$HTSmaxk_1$	Max it before granularity is removed (using Setting 1)	$2n/3$
HTS(+D)	$HTSmaxk_2$	Max it before granularity is removed (using Setting 2)	$2n$
(H)TS+D	Max_γ	Max it before diversification	$2n$

1. VRP and m-PSP with $m = 2, 3, 5, 6, 7$: Setting 1 (using $HTSmaxk_1$)
2. 4-PVRP: Setting 2 (using $HTSmaxk_2$)

As listed in Table 11.1, Settings 1 and 2 only differ in the value of the parameter *HTSmaxk* while all other parameters remain at a fixed value. Once set up as previously explained, each algorithm is run only once per instance. All algorithms are deterministic, but the results presented in the subsequent sections are aggregated per instance class to avoid extensive tables of results.

11.4.1 VRP and m-PSP

Table 11.2 summarizes the results of our algorithms for the VRP, on the four classes of instances A, B, P and vrpnc. Computational results show that the metaheuristics designed perform well on this particular problem because average gaps to optimality are around 0.80%. $HTS(+D)$ performs better than $TS(+D)$ on three of four instance classes and the hybridization lowers the average gap to optimality. $HTS + D$ results on the VRP can be further improved if the diversification procedure is activated a bit later on class A or sooner on class B: gap for A = 0.48% if $Max\_\gamma = 3n$ instead of $2n$, and gap for B = 0.89% if $Max\_\gamma = 3n/2$. As expected, the relaxed b-matching is computed very fast (0.28s) and it produces only a small number of edges (4%).

Table 11.3 shows the results of our algorithms for the m-PSP on Euclidean TSPLIB instances already used for assessing m-PSP algorithms in [7]. Our metaheuristics perform well on this problem because average gaps remain lower than 0.10%. *HTS* is the best algorithm, better than $HTS + D$, which means that our diversification procedure is used here too soon. The b-matching selects on average 15% of the edges.

Table 11.2 Results for the VRP (1-PVRP); m is the number of periods; NbI is the number of instances available; LB^* is the ratio between the best known lower and upper bounds, which is equal to 1 if both are optimal; Δ (resp. δ) is the average percentage deviation from the optimal solution value (resp. best known upper bound) for each instance class; σ is the standard deviation of Δ (resp. δ); s is the average duration in seconds to reach the best solution found; sBM is the average computing time of the linear b-matching, in seconds, to obtain the first set of edges for the b-matching granularity; and $Bm = NBm/TNe$ is the proportion of edges used by the linear b-matching solution, and used for composing the first set of edges for the granularity (NBm = number of edges used by the linear b-matching solution, TNe = total number of edges of the initial graph).

instance	m	NbI	LB^*	TS			$TS+D$			HTS			$HTS+D$		
class				Δ	σ	s	Δ	σ	s	Δ	σ	s	Δ	σ	s
A	1	27	1	0.56	0.76	3.28	**0.53**	0.73	2.80	**0.54**	0.50	2.88	**0.54**	0.57	3.43
B	1	23	1	**0.84**	1.47	1.97	0.95	1.49	2.46	0.96	1.50	3.29	0.93	1.45	3.77
P	1	23	1	0.50	0.56	3.63	0.56	0.61	2.91	0.47	0.54	3.04	**0.41**	0.53	3.45
vrpnc	1	7	-	1.49	1.71	12.69	**1.22**	1.52	25.02	**1.23**	1.37	26.02	1.26	1.85	17.76
Average		80	1	0.85	1.12	5.39	0.81	1.10	8.30	0.80	0.98	8.81	**0.78**	1.10	7.10

	m	Bm	sBm
A	1	0.05	0.07
B	1	0.05	0.08
P	1	0.06	0.09
vrpnc	1	0.02	0.88
Average		0.04	0.28

11.4.2 m-PVRP with $2 \leq m \leq 7$

Tables 11.4 to 11.7 summarize our results for the m-PVRP with $2 \leq m \leq 7$ on four classes of VRP instances: A, B, P and vrpnc. Two important preliminary remarks have to be made. First, when m increases, the number of instances

Table 11.3 Results for the m-PSP; for an explanation of the table entries, we refer to the caption of Table 11.2.

instance	m	NbI	$LB*$	TS		$TS+D$		HTS		$HTS+D$			
class				Δ	s	Δ	s	Δ	s	Δ	s	sBm	Bm
bays29	1	1	1	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	0.08	0.08
bays29	2	1	1	0.25	0.06	0.25	0.06	0.11	4.31	**0.09**	0.16	0.14	0.09
fri26	1	1	1	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	0.09	0.10
fri26	2	1	1	**0.00**	3.28	0.09	0.05	**0.00**	0.09	0.09	0.05	0.17	0.11
gr17	1	1	1	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.03	0.13	0.21
gr17	2	1	1	**0.08**	0.09	**0.08**	0.20	0.12	0.05	**0.08**	0.16	0.25	0.21
gr17	3	1	1	0.18	0.12	0.17	0.17	0.18	0.39	**0.09**	1.25	0.38	0.22
gr17	4	1	1	**0.00**	1.00	**0.00**	0.56	0.10	0.17	0.16	0.45	0.50	0.18
gr21	1	1	1	**0.00**	0.02	**0.00**	0.02	**0.00**	0.00	**0.00**	0.02	0.10	0.12
gr21	2	1	1	**0.00**	0.41	0.19	1.37	**0.00**	1.84	0.25	0.06	0.21	0.14
gr21	3	1	1	**0.02**	2.15	**0.02**	1.30	0.07	0.20	**0.02**	2.56	0.30	0.16
gr24	1	1	1	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	0.10	0.09
gr24	2	1	1	**0.00**	0.86	**0.00**	2.93	**0.00**	0.62	**0.00**	2.11	0.17	0.13
gr24	3	1	1	0.35	0.39	**0.25**	0.56	**0.27**	0.33	0.43	1.47	0.26	0.15
gr24	4	1	1	0.22	0.00	0.22	0.02	**0.11**	1.53	0.14	1.72	0.35	0.20
Average				0.07	0.56	0.08	0.48	**0.06**	0.64	0.09	0.67	0.22	0.15
σ				0.12		0.10		0.08		0.12			

Table 11.4 Results for the 2-PVRP; for an explanation of the table entries, we refer to the caption of Table 11.2.

instance	m	NbI	LB^*	TS			$TS+D$			HTS			$HTS+D$		
class				δ	σ	s	δ	σ	s	δ	σ	s	δ	σ	s
A	2	26	0.97	1.11	0.63	9.37	0.92	0.60	9.53	**0.81**	0.56	8.55	1.15	0.85	7.76
B	2	23	0.98	1.10	1.14	10.27	0.71	0.56	11.39	**0.68**	0.73	8.53	0.72	0.66	8.86
P	2	19	0.98	1.08	0.83	10.28	**1.03**	0.61	7.78	1.16	0.86	8.44	**1.02**	0.74	10.09
vrpnc	2	7	0.95	1.48	0.67	39.87	1.28	0.64	55.50	1.25	0.76	47.23	**1.03**	0.78	68.44
Average		75	0.97	1.20	0.82	17.45	**0.98**	0.60	21.05	**0.97**	0.73	18.20	**0.98**	0.76	23.79

	m	Bm	sBm
A	2	0.10	0.15
B	2	0.10	0.14
P	2	0.11	0.16
vrpnc	2	0.04	1.67
Average		0.09	0.53

available decreases because there are only n edges connected to the depot, and each route uses two of them. Second, gaps are computed from the best upper bound known. These are not proven to be optimal but LB^* gives an idea of their quality. LB^*=0.99% suggests that the best upper bound is very close to the optimal value. $LB* = 0.95\%$ means there is a 5% gap between the best known upper and lower bounds.

Figure 11.3 shows the evolution of the percentage deviation from the best known solutions over time for vrpnc instances. It suggests that the dominance of $HTS + D$ over the three other algorithms is reinforced when m increases. This remark is confirmed by most tables of results: $HTS(+D)$ is the best performing of the algorithms since it has the lowest gap from the best known upper bounds on most instances, except for those of 4-PVRP. The relaxed b-matching necessary is still computed very fast (four seconds for the vrpnc if $m = 5, 6, 7$) and the percentage of edges used is quite low (14% overall). $HTS+D$ results can be significantly improved if a specific setting of Max_γ is applied: e.g., overall average gap of $HTS + D$ on the 2-PVRP can be reduced from 0.98% to 0.91% if the diversification threshold Max_γ is slightly reduced from $2n$ to $3n/2$.

11.5 Conclusion

The partial tabu algorithm we designed gives good results not only on the m-Peripatetic Vehicle Routing Problem, but also on two well-known special cases: the VRP and the m-PSP. Its hybridization with the perfect b-matching through granularity improves significantly the algorithm efficiency, especially when it is adequately combined with the diversification procedure.

Table 11.5 Results for the 3-PVRP; for an explanation of the table entries, we refer to the caption of Table 11.2.

instance	m	NbI	LB^*	TS			$TS+D$			HTS			$HTS+D$		
class				δ	σ	s	δ	σ	s	δ	σ	s	δ	σ	s
A	3	25	0.98	1.28	1.20	14.14	1.05	0.79	12.97	1.07	0.88	11.30	**0.84**	0.85	13.77
B	3	22	0.98	1.94	1.91	15.51	1.12	0.91	17.27	1.47	1.35	14.51	**1.02**	0.98	14.86
P	3	14	0.99	0.97	0.45	14.23	**0.75**	0.33	13.13	0.88	0.49	0.17	**0.76**	0.36	14.08
vrpnc	3	7	0.95	1.06	0.54	69.59	**0.97**	0.76	85.80	1.13	0.76	67.50	**0.98**	0.68	47.44
Average		68	0.97	1.31	1.02	28.37	0.97	0.70	32.29	1.14	0.87	23.37	**0.90**	0.72	22.54

	m	Bm	sBm
A	3	0.15	0.20
B	3	0.15	0.20
P	3	0.17	0.23
vrpnc	3	0.07	2.44
Average		0.13	0.77

Table 11.6 Results for the 4-PVRP, for an explanation of the table entries, we refer to the caption of Table 11.2.

instance	m	NbI	LB^*	TS			$TS+D$			HTS			$HTS+D$		
class				δ	σ	s	δ	σ	s	δ	σ	s	δ	σ	s
P	4	8	0.99	**0.32**	0.24	19.28	0.37	0.33	11.32	0.43	0.19	12.33	0.45	0.20	4.75
vrpnc	4	6	0.96	0.69	0.31	111.30	0.56	0.30	104.92	**0.51**	0.34	158.50	**0.53**	0.29	54.92
Average		14	0.97	0.50	0.27	65.29	**0.46**	0.31	58.12	**0.47**	0.26	85.41	0.49	0.24	29.83

	m	Bm	sBm
P	4	0.28	0.32
vrpnc	4	0.09	3.65
Average		0.18	1.98

Table 11.7 Results for the m-PVRP with $m = 5, 6, 7$; for an explanation of the table entries, we refer to the caption of Table 11.2.

instance	m	NbI	LB^*	TS			$TS+D$			HTS			$HTS+D$		
class				δ	σ	s	δ	σ	s	δ	σ	s	δ	σ	s
P	5,6,7	11	0.99	0.40	0.24	63.84	0.44	0.31	55.04	**0.34**	0.24	61.18	0.40	0.24	37.53
vrpnc	5,6,7	10	0.96	1.08	0.92	208.73	0.76	0.43	187.46	0.57	0.47	235.14	**0.50**	0.46	181.40
Average		21	0.97	0.69	0.58	168.20	0.60	0.37	121.25	**0.45**	0.35	148.16	**0.45**	0.35	109.40

	m	Bm	sBm
P	4	0.22	0.94
vrpnc	4	0.11	4.17
Average		0.16	2.55

References

1. P. Augerat. *Approche polyédrale du problème de tournées de véhicules*. PhD thesis, Institut National Polytechnique de Grenoble, France, 1995.
2. R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385, 2008.
3. N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and L. Sandi, editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, UK, 1979.

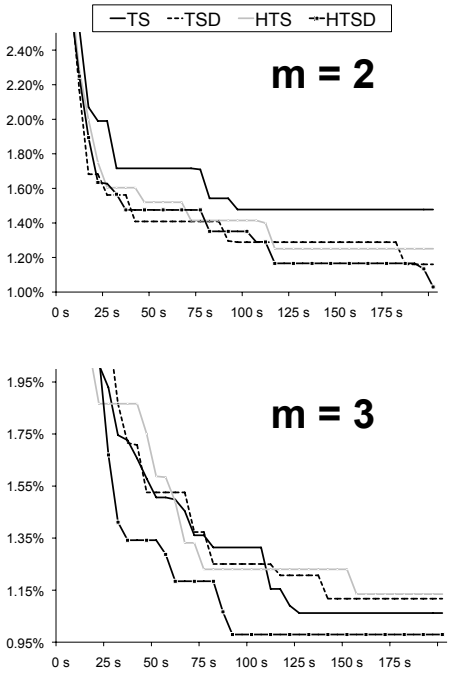


Fig. 11.3 Evolution of δ over time (in seconds) on the vrpnc instances.

4. G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.
5. J.F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.S. Sormany. New heuristics for the vehicle routing problem. In A. Langevin and D. Riopel, editors, *Logistics systems: design and optimization*, pages 279–298. Wiley, 2005.
6. J.B.J.M. De Kort and A. Volgenant. On the generalized peripatetic salesman problem. *European Journal of Operational Research*, 73:175–180, 1994.
7. E. Duchenne, G. Laporte, and F. Semet. Branch and cut algorithms for the undirected m-peripatetic salesman problem. *European Journal of Operational Research*, 162:700–712, 2005.
8. E. Duchenne, G. Laporte, and F. Semet. The undirected m-peripatetic salesman problem: Polyhedral results and new algorithms. *Operations Research*, 55(5):949–965, 2007.
9. J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.
10. F. Glover and M. Laguna. Tabu search. In C.R. Reeves, editor, *Modern heuristic techniques for combinatorial problems*, pages 70–150. Blackwell, Oxford, UK, 1993.
11. J. Krarup. The peripatetic salesman and some related unsolved problems. In B. Roy, editor, *Combinatorial Programming Methods and Applications*, pages 173–178. Reidel, Dordrecht, 1975.
12. S.U. Ngueveu, C. Prins, and R. Wolfler Calvo. Bornes supérieures et inférieures pour le problème de tournées de véhicules m-péripatétiques. In *Actes de la 7ème conférence internationale de Modélisation et Simulation (MOSIM)*, volume 3, pages 1617–1625, Paris, France, 2008.

13. E.D. Taillard. Parallel iterative search methods for vehicle routing problems. Technical Report G-2002-15, Les Cahiers du GERAD, Canada, 2002.
14. P. Toth and D. Vigo. *The vehicle routing problem.* SIAM, Philadelphia, 2002.
15. P. Toth and D. Vigo. The granular tabu search and its application to the vehicle routing problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003.
16. R. Wolfler Calvo and R. Cordone. A heuristic approach to the overnight security service problem. *Computers & Operations Research*, 30:1269–1287, 2003.

Index

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7,