# Chapter 4
# Heuristics and Metaheuristics for Fixed-Charge Network Design


**Teodor Gabriel Crainic and Michel Gendreau**

## 1 Introduction

While the methods based on exact solution principles described in Chap. 3 provide means to solve network design problem instances that have become of significant size over the years, it is important to realize that they are not the only way by which one can address network design problems. Actually, over the last 60 years or so, various approaches have been proposed to derive approximate solutions for various types of network design problems. These approaches have been used mostly to deal with the larger instances that are typically encountered in the context of real-life applications, which cannot be addressed by exact methods. In some cases, constraints on the time available for deriving good feasible solutions to complex problems has been a strong incentive for resorting to heuristics and metaheuristics.

Heuristic and metaheuristic approaches differ from the mathematical-programming based methods of Chap. 3 in many ways. A key difference is that they are based on the exploration of a *search space*, which is often quite different from the notion of feasible space of exact approaches. We will examine the notion of search space more thoroughly in the next section, as well as other important concepts.

Approximate approaches to network design problems can be broken down into three large classes: first, so-called *classical heuristics*, which rely on fairly simple

T. G. Crainic
CIRRELT and AOTI, Université du Québec à Montréal, Montréal, QC, Canada
e-mail: TeodorGabriel.Crainic@cirrelt.net

M. Gendreau (✉)
CIRRELT and MAGI, Polytechnique Montréal, Montréal, QC, Canada
e-mail: Michel.Gendreau@cirrelt.net



T. G. Crainic et al. (eds.), *Network Design with Applications to Transportation and Logistics*, https://doi.org/10.1007/978-3-030-64018-7_4

rules for building and improving tentative solutions; second, *metaheuristics*, which are methods that rely on sophisticated search strategies to derive very good (often near-optimal) solutions to the problem at hand; and third, *matheuristics*, which combine algorithmic components from metaheuristics with procedures derived from exact methods applied to the model formulation. In this chapter, we will examine these three classes, as well as parallel methaheuristics, which are methods that leverage the power of parallel computing to find better approximate solutions.

This chapter is organized as follows. In Sect. 2, we first present a number of basic concepts that are central to the way in which heuristics and metaheuristics tackle combinatorial problems. Section 3 is devoted to the more traditional heuristics, while the two following sections are devoted to the two main classes of metaheuristics, i.e., *neighborhood-based* metaheuristics and *population-based* metaheuristics. The first class refers to methods that follow a trajectory of solutions, moving at each step from a current tentative solution to a different neighbor. Population-based methods rely on the the application of various operations on a population of several possible solutions to the problem at hand with the objective of identifying an interesting one at the end of the procedure; in several of these methods, the operations considered mimick processes that are observed in the evolution of species. Some important applications of matheuristics to network design problem are then presented in Sect. 6. Section 7 follows and presents solution approaches that use parallel computing; while these methods often combine ideas and elements from the previous sections, several go much further and are among the best meta- or matheuristics for network design problems. Bibliographical notes are provided in Sect. 8. We summarize the main conclusions of this chapter and provide some research perspectives in Sect. 9.

In Sects. 3–5, we review in different subsections several methods that fall under the general heading of the section. For each of these methods, we first briefly recall the general principles of the method. This is followed in most cases by a presentation of the application of the method to one or several of the network design problems described in Chap. 2. In some cases, when we are not aware of any direct application of a method to network design problems in transportation and logistics, the presentation of relevant applications is deferred to Sects. 6 or 8.

## 2 Basic Concepts

We first examine the notion of *search space*, which is central in heuristics and metaheuristics. Furthermore, the exploration of this search space is conducted normally by considering *neighborhoods* or *populations* of solutions. We review these basic concepts, as well as a few others, in the following.

## 2.1 Search Space

The *search space* that will be explored by the search is a key component of the methods examined in this chapter. This notion refers to the space of solutions that can be considered through the exploration. While it may seem natural to equate the search space with the set of feasible solutions to a problem, there are many situations where this is not attractive.

To better understand this, consider the single-commodity fixed-charge transportation problem (FCTP) and the formulation proposed in Sect. 2.3 of Chap. 2. This problem is defined on a network $\mathscr{G} = \{\mathscr{N}, \mathscr{A}\}$, the set $\mathscr{A}$ encompassing the possible design arcs, between the origin (source) and destination (sink) nodes of set $\mathscr{N} = \mathscr{N}^o \cup \mathscr{N}^d$, among which the selection is to be made. For simplicity sake, we assume in the following that the underlying bipartite graph is complete. Each design arc is characterized by a *fixed* selection cost, $f_{ij}$, a *unit* flow *transportation cost*, $c_{ij}$, and a *capacity*, $u_{ij}$, limiting the volume of commodity one may assign to the arc. The objective is to fulfill at minimal total cost, computed as the sum on the total fixed and transportation costs, the demand for transportation between the origins, each with *supply* (*availability*) $w_i > 0$ of the given commodity, and destinations, each with a *demand* (*request*) $w_i < 0$ of the same commodity.

Two types of decision variables are defined: $x_{ij}$ continuous variables that refer to the amount of flow going from vertex $i$ to vertex $j$, and $y_{ij}$ binary variables that indicate whether arc $(i, j)$ is used in the solution or not. Thus, one could imagine that the search space would consist of feasible pairs of $(\mathbf{x}, \mathbf{y})$ vectors. However, one can do much better because of the close relationships between the $\mathbf{x}$ and $\mathbf{y}$ vectors that make up any optimal solution. In particular, in an optimal solution, one will have $y_{ij} = 1$ *iff* $x_{ij} > 0$. One could thus define the search space with respect to the $\mathbf{x}$ variables only. An alternate approach, which is much more easily implemented, defines the search space with respect to the $\mathbf{y}$ variables. For any given value $\bar{\mathbf{y}}$ of the $\mathbf{y}$ vector, a complete solution can be recovered by solving an auxiliary transportation problem:

$$\text{Minimize} \quad \sum_{(i,j)\in\mathscr{A}} c_{ij}x_{ij} \tag{4.1}$$

$$\text{Subject to} \quad \sum_{j\in\mathscr{N}_i^+} x_{ij} = |w_i|, \quad \forall\, i \in \mathscr{N}^o, \tag{4.2}$$

$$\sum_{j\in\mathscr{N}_i^-} x_{ji} = |w_i|, \quad \forall\, i \in \mathscr{N}^d, \tag{4.3}$$

$$x_{ij} \le u_{ij}\bar{y}_{ij}, \qquad \forall\,(i, j) \in \mathscr{A}, \tag{4.4}$$

$$x_{ij} \ge 0, \qquad \forall\,(i, j) \in \mathscr{A}. \tag{4.5}$$

where constraints (4.2) and (4.3) enforce the supply ($w_i, i \in \mathscr{N}^o$) and demand ($w_i, i \in \mathscr{N}^d$) conditions at origin and destination nodes, respectively, while constraint (4.4) limits the flow on each selected arc to its capacity, $u_{ij}$.

It is important to note that, in many methods, the search space is often not limited to feasible solutions. In fact, many extremely successful metaheuristics rely heavily on the possibility of considering infeasible solutions during the search, especially when dealing with very constrained problems. This is usually managed by adding penalties to the objective to account for violations of some constraints (see Gendreau et al. 1994, for a very successful use of these ideas).

## 2.2 Neighborhoods

Several of the methods that we discuss in this chapter, notably, Local Search improvement heuristics and neighborhood-based metaheuristics, rely heavily upon the concept of *move*, which refers to the transition from one solution in the search space to another by the application of some transformation. In many methods, these transformations are rather limited and give rise to solutions that are just slightly modified, but this is not always the case and moves implying major changes to the solutions were proposed for several methods.

The set of all the solutions that can be produced by applying an allowable move to a given solution $S$ gives rise to the *neighborhood* $\mathcal{N}(S)$ of this solution $S$. The definition of the neighborhood structures (there could be one or several depending on the complexity of the method) that will be used in a search process is one of its central features. The definition of these neighborhood structures is highly dependent on the choices made for the search space, since neighborhoods allow us to go from one element in the search space to another (or, more correctly, define a set of other elements of the search space that one could reach in a single move).

Consider again the fixed-charge transportation problem and suppose that we have chosen to explore the space of the binary $y$ variables. In this search space, a neighborhood structure that is commonly used is the so-called *add-drop neighborhood*, which associates to any given $\bar{\mathbf{y}}$ vector the set of all $\mathbf{y}$ vectors that differ from $\bar{\mathbf{y}}$ in only a single entry (either we *add* an arc $(i', j')$ for which we had previously $\bar{y}_{i'j'} = 0$, or we drop an arc for which $\bar{y}_{i'j'} = 1$). Thus the add-drop neighborhood $N_{AD}(\bar{\mathbf{y}})$ is the set $\{\mathbf{y} | y_{ij} = \bar{y}_{ij}, (i, j) \in \mathcal{A} \setminus (i', j')$ and $y_{i'j'} = 1 - \bar{y}_{i'j'}, \forall (i', j') \in \mathcal{A}\}$.

An alternate neighborhood structure for the same search space is that of the *swap neighborhood*, which corresponds to changing simultaneously two entries in the $\bar{\mathbf{y}}$ vector: a null one that now takes the value 1 and a positive one that becomes 0.

Neighborhood-based methods usually require that, at each iteration, the solutions in the neighborhood $\mathcal{N}(S)$ of the current solution $S$ be examined to find an attractive candidate solution for the next iteration. There are several ways of performing this exploration. A very common one consists in examining all the elements of $\mathcal{N}(S)$ and in selecting the one with the best objective function value. This is called *best-improvement* search. If neighborhoods are large, one could instead stop the procedure as soon as a solution that improves on $S$ is found, which is *first-improvement* search. More sophisticated ways to perform the evaluation in

the neighborhood $\mathcal{N}(S)$ have been proposed, mostly to reduce the computational burden of each iteration. These are often refereed to as *candidate-list strategies*.

## 2.3 Populations

As their name implies, population-based metaheuristics rely on the exploitation of populations (i.e., samples) of solutions of the problem at hand to explore a chosen search space. Usually, most of these solutions are feasible ones, but in some cases, infeasible solutions may be considered as well. Central to population-based methods is the process of constructing new solutions by combining elements or features of existing ones.

There is a wide range of population-based metaheuristics and the specific mechanisms that they use thus vary quite a lot, but any population-based metaheuristic must address a number of fundamental questions:

- How is the initial population created?
- At each iteration, how are chosen existing population members to create new ones? (*selection*)
- How are features from selected population members selected? (*crossover* or other combination procedure)
- What other modifications are performed on elements from the population? (*mutation* or *education*)
- How are less interesting members of the population deleted?
- When should the search process stop?

We will see in Sect. 5 the answers provided by different methods to these questions.

## 2.4 Evaluating the Performance of Heuristics and Metaheuristics

In general, the evaluation of the performance of heuristics and metaheuristics is a much more complex issue than for exact methods. Among other things, it is not just a matter of measuring how much computing (CPU) time a method requires to find the optimal solution to a problem. Furthermore, except in very special cases, there are very few theoretical results that one can derive in this area. The evaluation of heuristics and metaheuristics thus relies heavily on empirical computational experiments performed on suitable sets of benchmark instances. Such benchmarks should be reasonably representative of the types of actual instances that one would like to solve with the methods at hand.

One must also avoid the pitfall of using the same sets of instances to calibrate the parameters of the methods being tested and to derive performance measures.

If this is not done, one may end up in a situation where the parameter values are too well adapted to the instances at hand (i.e., this is a case of *over-fitting*) with the consequence that the performance observed on the benchmark instances would not generalize to other instances, thus leading to completely erroneous conclusions. Ideally, one would also like to have in benchmark instances that are small enough to be tackled by exact methods to allow for the estimation of the optimality gaps between heuristic solutions and actual optimal solutions.

Another issue that is far from trivial is deciding how much time or how many iterations to allow to heuristics, since many of these methods do not have obvious a priori stopping criteria. With additional time, the quality of solutions improves, but often in a very different fashion. Different methods thus display different *performance profiles* over time, which makes a direct comparison often rather difficult.

A factor that often complicates comparisons between various methods aimed at solving the same class of problems is the fact that different authors use different sets of test instances, which makes it almost impossible to derive any meaningful comparisons between the methods. We are not aware of any widely used benchmark for the fixed-charge transportation problem. The situation is different for the multicommodity capacitated fixed-charge network design problem (MCFND). Most authors who proposed solution methods for the MCFND have relied on the same set of benchmark instances to assess the performance of this method. This benchmark was proposed in Gendron and Crainic (1994), and it is made up of two sets of instances. The main set, which is called set **C**, is made up of 43 instances having between 20 and 100 nodes, between 100 and 700 arcs, from 10 to 400 commodities, different fixed to variable costs ratios, and loose or tight capacity constraints, providing the means to thoroughly assess the performance of algorithms on a wide range of problem features; this is the set used by most authors in their computational experiments. A second set, called set **R**, was created to study more systematically the impact of problem characteristics on algorithmic performance. It is made up of 18 basic networks, having from 10 to 20 nodes, 25 to 300 arcs, and 10 to 200 commodities; for each network, 9 instances are created by considering three levels of fixed cost and three levels of capacity tightness, thus yielding a total of 162 instances. In this chapter, we will refer to these sets of benchmark instances as sets **C** and **R** of the Gendron-Crainic benchmark.

## 3 Classical Heuristics

When dealing with large or difficult problems in combinatorial optimization, heuristic methods have been used since the beginnings of operations research. We refer to methods used since the early ages of operations research as *classical* ones. In general, one distinguishes between constructive and improvement heuristics, but these are often used in conjunction to provide better-quality solutions.

## 3.1 Constructive Heuristics

The main objective pursued in constructive heuristics is simply to obtain some feasible solution for the problem at hand, with the hope that this solution will be "good enough" for the intended usage, or that it can serve as initial solution for a more involved heuristic procedure. The archetypical example of a constructive heuristic is the so-called *greedy heuristic*, which builds a solution one element at the time, at the lowest cost possible (assuming that the problem is a minimization one).

To illustrate how such a method works, let us consider the fixed-charge transportation formulation recalled in Sect. 2 (see also Sect. 2.3 of Chap. 2). One way of applying a greedy approach for this problem is to compute the unit linearized cost $c'_{ij} = f_{ij}/u_{ij} + c_{ij}$ for all arcs $(i, j) \in \mathscr{A}$, and to select the arc with the minimum value. One then assigns a flow of value $x_{ij} = \min\{|w_i|, |w_j|\}$ to arc $(i, j)$. The values for $w_i$ and $w_j$ are then decreased by $x_{ij}$, since the transportation requests for nodes $i$ and $j$ have been partially fulfilled. The process then goes through another iteration, without taking into account the arc $(i, j)$, which can no longer be used, until all requests have been assigned. When the problem is defined on a complete bipartite graph and when demands are balanced, i.e., when $\sum_{i \in \mathscr{N}^o} w_i = \sum_{j \in \mathscr{N}^d} w_j$, this process leads to a feasible solution.

Greedy methods are fast but yield results of dubious quality in many cases, particularly for network design problems. It has been experimentally shown, for example, that the last arcs selected by the greedy heuristic above for the transportation problem are among the most costly ones in the network. A constructive method alleviating such shortcomings for network design makes use of the same unit linearized arc cost defined above, but is based on solving the linear programming relaxation of the problem with those costs, and selecting the arcs with positive flow in the optimal solution. More precisely, one first solves the minimum cost transportation problem (4.1)–(4.5) with $c_{ij} = c'_{ij}$. Then, $y_{ij} = 1$ if $x_{ij} > 0$ in the optimal solution, and 0, otherwise. This procedure has experimentally produced very good feasible solutions not only for the fixed-charge transportation problem, but also more broadly for the single and the multicommodity capacitated fixed-charge network design formulations.

## 3.2 Improvement Methods (Local Search)

Improvement methods are the natural complement to constructive heuristics, in the sense that these are methods that are meant to produce sequences of feasible solutions that improve on one another with respect to the objective function of the problem at hand. Most of these methods are based on the principles of *Local Search*, which iteratively applies "local" modifications to a so-called *current solution* in such a way that the modified solution improves the objective value at each step. This

monotonicity condition ensures that the method will not cycle. It will eventually reach a solution such that no improving one can be determined by the application of the possible local modifications. Whenever this happens, the search has found a *local optimum* and terminates.

It must be emphasized that Local Search methods rely heavily rely on the concepts of *search space* and *neighborhoods* presented in Sect. 2. It must be noted, however, that traditional Local Search methods rarely consider infeasible solutions in their exploration process.

### 3.2.1 Basic Local Search

The basic principle of Local Search is simply to explore in an iterative fashion the selected search space for the problem at hand by performing at each iteration a move from the *current solution S* to an improving feasible one selected in its neighborhood $\mathcal{N}(S)$, until a local optimum is encountered. The improving solution chosen at each iteration can be selected according to best-improvement or first-improvement rules.

### 3.2.2 A Local Approach Search for the Fixed-Charge Transportation Problem

We now describe a fairly simple, yet effective, approach for the fixed-charge transportation problem. This Local Search method relies on the exploration of the extreme points of the polyhedron of the transportation problem defined by equations

$$\sum_{j \in \mathcal{N}_i^+} x_{ij} = |w_i|, \quad \forall\, i \in \mathcal{N}^o, \tag{4.6}$$

$$\sum_{j \in \mathcal{N}_i^-} x_{ji} = |w_i|, \quad \forall\, i \in \mathcal{N}^d, \tag{4.7}$$

$$x_{ij} \leq u_{ij}, \qquad \forall\, (i, j) \in \mathcal{A}, \tag{4.8}$$

$$x_{ij} \geq 0, \qquad \forall\, (i, j) \in \mathcal{A}. \tag{4.9}$$

These extreme points correspond to feasible basic solutions of the transportation problem. It has been known since the 1950s (Hirsch and Dantzig 1954), that an optimal solution of the fixed-charge transportation problem can be found in one of the extreme points of this polyhedron. Thus, one can simply explore the search space defined by these extreme points (feasible bases of the system (4.6)–(4.9)). A natural way to do so is to consider *pivot* operations from one feasible basic solution to another, and use them to define a neighborhood structure on this search space. It is also interesting to note that, since bases of the system (4.6)–(4.9) correspond to spanning trees of the underlying bipartite graph, one can also interpret the search using this neighborhood as searching the space of the spanning trees of the bipartite graph, where adjacent spanning trees only differ in two edges.

A Local Search procedure for the fixed-charge transportation problem can thus start from any basic solution to (4.6)–(4.9), and proceed by performing pivots on this system. Let's note $\bar{\mathbf{x}}$ the vector of continuous variables corresponding to the current extreme point solution. The cost of this solution is then easily obtained as the sum of the variable $\sum_{(i,j)\in\mathscr{A}} c_{ij}x_{ij}$ and fixed costs $f_{ij}$ of arcs $(i, j)$ for which $x_{ij} > 0$.

This basic Local Search can be expected to rapidly run into a local optimum and thus to terminate prematurely. One way to improve its performance consists in considering, when running into a local optimum, an extended neighborhood that contains extreme points that are two pivots away from the current solution (i.e., solutions that are obtained by pivoting into the current basis a pair of non-basic variables). Obviously, the exploration of the set of extreme points that are two pivots away is significantly more expensive than just looking at adjacent extreme points, but it yields significantly better solutions. Defining several such neighborhoods, e.g., 1-pivot, 2-pivots, 3-pivots, ..., and rules specifying when and how the search passes from one neighborhood to another leads to metaheuristics.

## 4 Neighborhood-Based Metaheuristics

As we already mentioned, neighborhood-based metaheuristics rely on following a trajectory of related solutions to the problem at hand; hence, they are also referred to as *trajectory-based methods*. In many ways, these methods can be seen as generalizations of the Local Search methods of the previous section, using the same basic concepts of search spaces and neighborhoods, with a few twists that we now explain.

All neighborhood-based metaheuristics rely on more sophisticated search strategies than Local Search, in particular by including more than one search heuristic. In this way, they perfectly illustrate the definition of metaheuristics (and matheuristics) as *heuristics guiding other heuristics*. Moreover, in the improvement methods of the previous section, one normally chooses the feasible neighbor with the best objective function value to become the new current solution and the search terminates whenever $\mathscr{N}(S)$ does not contain any improving feasible neighbor. This is not the case in neighborhood-based metaheuristics, which often use different rules for selecting the new current solution and do not terminate when they encounter local optima.

We first focus on Tabu Search methods, which have seen very successful applications to classical network design problems. We then present rapidly four other neighborhood-based metaheuristics, which have been integrated into hybrid methods or matheuristics, or applied to complex network design problems that are beyond the scope of this chapter: Simulated Annealing, Iterated Local Search, Greedy Randomized Adaptive Search Procedure, and Variable Neighborhood Search.

## 4.1 Tabu Search

Traditional Local Search methods are plagued by the fact that their exploration of the search space terminates whenever they run out of improving neighbors, i.e., whenever they run into a *local optimum* of the problem with respect to the chosen neighborhood structure. In many problems, this can lead to very poor heuristic solutions. The key ideas of Tabu Search is (1) to continue exploring the search space even when a local optimum is encountered, and (2) to *learn* during the exploration about the search space, the trajectory, and the search behavior (e.g., identifying variables or zones critical to good solutions). Continuing the exploration beyond a local optimum may, however, easily lead to cycling: after moving from a local optimum to one of its neighbors, the search could very well move back to the just-visited local optimum. To prevent this from happening, one must forbid some moves by declaring then *tabu* for a certain time length or number of iterations. In practice, one records in some *short-term memory* key information on the moves that were performed in the most recent iterations and declares tabu either the reverse moves or moves involving particular configurations of the solution attributes (e.g., paths for commodities recently involved in moves). Note that the tabu status may be lifted if the candidate solution is *improving* with respect to an *aspiration criterion*. The latter is generally defined as a minimum threshold by which the candidate solution must better than the local or global current best solution (e.g., the objective-function value of the candidate must be lower by at least $\alpha\%$ than the value of the current best), in order for the tabu status to be lifted.

Actually, Tabu Search goes well beyond simply exploiting short-term memory. Longer-term memories are used to implement two key ideas: *search intensification* and *search diversification*.

The idea behind search intensification is, from times to times, to interrupt the regular search process to explore more thoroughly portions of the search space in which good solutions, e.g., the best known solution, have been encountered. This more thorough search is often accomplished by switching to a different neighborhood structure: for instance, if one uses an add-drop neighborhood structure for regular search, then intensification could use a swap neighborhood.

Diversification can be seen as the complement of intensification. Here, the main objective is to insure that the search covers a wide portion of the search space, i.e., that several different possible solutions are examined. Diversification is often based on long-term memories, such as *frequency memory*, which records the number of times that some solution element, e.g., a design arc, has appeared in the current solution. The basic idea is to implement algorithmic mechanisms to "force" these less frequent elements into the current solution, thus redirecting the search to unexplored or little explored parts of the search space. Diversification has become a key component of most meta- and matheuristic strategies.

### 4.1.1 Tabu Search for the Fixed-Charge Transportation Problem

A natural way of applying Tabu Search to the fixed-charge transportation problem is by extending the ideas of Sect. 3.2.2 with respect to the choice of search space and neighborhood structure. Thus, one can implement a Tabu Search metaheuristic using the set of extreme points of the polyhedron defined by the system (4.6)–(4.9) as search space, and a neighborhood structure defined by pivot operations. Because of the more sophisticated search mechanisms available in Tabu Search, one can just consider 1-pivot moves and adjacent extreme points as neighbors of a given solution.

The search can be initiated from any feasible solution to (4.6)–(4.9). Since moves correspond to pivot operations, they can be defined in terms of flow (continuous) variables entering the basis, or equivalently, the arcs of the spanning tree to which this basis corresponds. In that context, tabu restrictions are defined with respect to the arcs of that spanning tree: (1) an arc leaving the spanning tree and becoming non-basic is prevented from reentering the basis for $\xi_{in}$ iterations; (2) when an arc becomes basic, it must remain in the solution for at least $\xi_{out}$ iterations. The values for these *tabu tenures* can be fixed throughout the search or be reset at some random values in a given interval during the search. These tabus prevent some solutions in the neighborhood of the current solution to be considered as candidates for the next move, unless these solutions are improving with respect to an aspiration criterion.

To reduce the computational burden, one can consider at each iteration only the moves involving non-basic arcs originating from a single origin. In the next iteration, the selected origin is moved to the next one. This *candidate list strategy* is disabled when all moves considered are tabu.

When the basic search process starts to stagnate (i.e., a large number of iterations have been performed without improving the best solution found), it is a good idea to consider intensifying the search in the hope of identifying better solutions. One way of inducing intensification is to reduce the impact of the fixed cost of arcs that have been present in the basis of the current solution for a large number of iterations. By reducing the contribution of the fixed cost of these arcs in the objective, these arcs become more attractive and thus more likely to be again present in the spanning tree. This reduction of fixed cost is applied until an overall better solution is found or for a certain number of moves.

Diversification can be encouraged in a fashion similar to the one used for intensification, but instead reducing the impact of the fixed cost of arcs that have seldom been present in the basis of the current solution. The length of this diversification phase can be fixed. Another option for diversification is to directly force into the spanning tree arcs that have been out of the basis for the longest time.

Because they do not stop at local optima, Tabu Search heuristics could, in theory, go on for ever. We thus need stopping criteria. In its simplest form, these could be a pre-defined number of iterations or, more usually, a number of iterations without improvement of the best solution found. In the context of a complex search strategy involving both intensification and diversification phases, it makes more sense to stop the search after a given number of such phases and return the best solution found.

### 4.1.2 Tabu Search for the Multicommodity Capacitated Fixed-Charge Network Design Problem

We now examine how Tabu Search can be used to tackle multicommodity capacitated fixed-charge network design (MCFND) formulations (Sect. 3, Chap. 3). We briefly recall the notation of the MCFND. The problem is defined on a network $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ shared by several commodities represented by set $\mathscr{K}$. The demand $w_i^k$ to satisfy for any commodity $k \in \mathscr{K}$ is defined at each node $i \in \mathscr{N}$ and, for simplicity, we assume that $\sum_{i \in \mathscr{N}} w_i^k = 0, k \in \mathscr{K}$, i.e., the demand is balanced for each commodity. Each potential design arc $(i, j) \in \mathscr{A}$ is characterized by its fixed cost $f_{ij}$, charged whenever the arc is included in the optimal design, its capacity $u_{ij}$, limiting the total flow of all commodities on the arc, and commodity-specific unit transportation costs $c_{ij}^k, k \in \mathscr{K}$. The MCFND minimizes the sum of the costs to select the arcs to be included in the design and the transportation costs to move the commodity flows, within the transportation capacities of the selected arcs. Using binary design variables $y_{ij}, (i, j) \in \mathscr{A}$, and continuous multicommodity flow variables $x_{ij}^k \geq 0, (i, j) \in \mathscr{A}, k \in \mathscr{K}$, the arc-based MCFND model is written as

$$\text{Minimize} \quad \sum_{(i,j) \in \mathscr{A}} f_{ij} y_{ij} + \sum_{k \in \mathscr{K}} \sum_{(i,j) \in \mathscr{A}} c_{ij}^k x_{ij}^k \tag{4.10}$$

$$\text{Subject to} \quad \sum_{j \in \mathscr{N}_i^+} x_{ij}^k - \sum_{j \in \mathscr{N}_i^-} x_{ji}^k = w_i^k, \ \forall i \in \mathscr{N}, \forall k \in \mathscr{K}, \tag{4.11}$$

$$\sum_{k \in \mathscr{K}} x_{ij}^k \leq u_{ij} y_{ij}, \qquad \forall (i, j) \in \mathscr{A}, \tag{4.12}$$

$$x_{ij}^k \geq 0, \qquad \forall (i, j) \in \mathscr{A}, \forall k \in \mathscr{K}, \tag{4.13}$$

$$y_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in \mathscr{A}. \tag{4.14}$$

Tabu Search has been applied quite successfully to complex facility location problems, which display strong similarities with network design problems (in fact, many facility location problems can be easily transformed into equivalent network design problems). Several of these successful implementations were based on the exploration of the search space of binary location (i.e., design) variables using the add-drop and swap neighborhood structures described in Sect. 2.2. This type of approach would thus seem attractive for tackling the MCFND: once design variables have been set to given values given by the vector $\bar{\mathbf{y}}$, corresponding continuous flow variables can be easily recovered by solving the auxiliary *minimum cost multicommodity network flow* (MCMNF) problem:

$$\text{Minimize} \quad \sum_{k \in \mathscr{K}} \sum_{(i,j) \in \mathscr{A}} c_{ij}^k x_{ij}^k \tag{4.15}$$

$$\text{Subject to} \quad \sum_{j \in \mathscr{N}_i^+} x_{ij}^k - \sum_{j \in \mathscr{N}_i^-} x_{ji}^k = w_i^k, \ \forall i \in \mathscr{N}, \forall k \in \mathscr{K}, \tag{4.16}$$

$$\sum_{k\in\mathscr{K}} x_{ij}^k \le u_{ij}\bar{y}_{ij}, \qquad \forall\,(i,j)\in\mathscr{A}, \tag{4.17}$$

$$x_{ij}^k \ge 0, \qquad \forall\,(i,j)\in\mathscr{A},\ \forall\,k\in\mathscr{K}, \tag{4.18}$$

$$y_{ij}\in\{0,1\}, \qquad \forall\,(i,j)\in\mathscr{A}. \tag{4.19}$$

Unfortunately, this fairly straightforward approach performs very poorly for network design, which presents a more complex network structure compared to location. Moves that close arcs may have a strong impact on the current solution, when closing paths that are used by many commodities between several origin-destination pairs, but moves that open new arcs are very often ineffective, since, in most cases, they will not lead directly to the creation of new paths for the commodities.

### Pivot-Based Neighborhoods

The ideas underlying the solution methods presented in Sects. 3.2.2 and 4.1.1 offer more promising alternatives. Here again, the key idea is to consider as search space the set of extreme points of the MCMNF problem, with all arcs open, i.e., the problem obtained by deleting the $\bar{y}_{ij}$'s from the linear program (4.15)–(4.18). This search space can be explored by performing pivots from one feasible solution to another. The original $c_{ij}^k$ arcs costs may be used or the linearized ones, i.e., $c_{ij}'^k = f_{ij}/u_{ij} + c_{ij}^k$, $(i,j)\in\mathscr{A}, k\in\mathscr{K}$. One generally expects to obtained better results by resorting to the linearized costs. As for the values of the design variables $y_{ij}$, they are easily recovered by setting $y_{ij} = 1$, when $\sum_{k\in\mathscr{K}} x_{ij}^k > 0$, and 0, otherwise.

At this point, it is important to recall that large MCMNF problems are solved more effectively using path-flow formulations and column generation techniques. This suggests switching to the path-based formulation and exploring the extreme points of the linear capacitated multicommodity flow subproblem of this formulation. Let $\mathscr{P}^k$ be the set of feasible paths in $\mathscr{G}$ for commodity $k\in\mathscr{K}$, let's define the decision variable $h_p^k$ as the volume of commodity $k$ moved on path $p\in\mathscr{P}^k$.

$$\text{Minimize} \sum_{(i,j)\in\mathscr{A}}\sum_{k\in\mathscr{K}}\sum_{p\in\mathscr{P}^k} c_{ij}^k\delta_{ij}^p h_p^k \tag{4.20}$$

$$\text{Subject to} \quad \sum_{p\in\mathscr{P}^k} h_p^k = d^k, \qquad \forall\,k\in\mathscr{K}, \tag{4.21}$$

$$\sum_{k\in\mathscr{K}}\sum_{p\in\mathscr{P}^k}\delta_{ij}^p h_p^k \le u_{ij}, \quad \forall\,(i,j)\in\mathscr{A}, \tag{4.22}$$

$$h_p^k \ge 0, \qquad \forall\,k\in\mathscr{K},\ \forall\,p\in\mathscr{P}^k, \tag{4.23}$$

where $\delta_{ij}^p$ indicates whether (i.e., $\delta_{ij}^p = 1$) or not (i.e., $\delta_{ij}^p = 0$) arc $(i, j) \in \mathscr{A}$ belongs to path $p \in \cup_{k \in \mathscr{K}} \mathscr{P}^k$ (thus, $x_{ij}^k = \sum_{p \in \mathscr{P}^k} \delta_{ij}^p h_p^k, \ \forall (i, j) \in \mathscr{A}$).

In this formulation, pivots are made with respect to the path variables $h_p^k$ (again, path costs may be the original or the linearized ones). A move thus corresponds to entering a currently unused path-flow variable $h_{p'}^{k'}$, $k' \in \mathscr{K}$, in the basis, which, in turn, forces out of the basis a path $p''$, of any commodity, for which the flow is driven to zero. Moves are evaluated by computing the difference in the total cost of the current solution, which is the sum of the variation of the total cost for the continuous variables plus the difference in the fixed cost of the binary design variables whose status has changed from open to closed or vice-versa. To prevent cycling, tabus are imposed on pivoting variables out of the basis, by assigning a random tabu tenure to each path entering the basis.

As in any path-based formulation, the number of paths can be huge. One must thus start the algorithm with a limited number of paths for each commodity and resort to column generation to identify additional ones. In this tabu search procedure, a column generation phase could be performed, for example, after a given number of moves without observing an improvement in the best known solution. As for search diversification, it could be performed after a set number of column generation phases without improvement of the objective. It is induced by closing a small subset of often-used arcs (as recorded in frequency memories) for some time.

### A Better Approach: Using Cycle-Based Neighborhoods

While the Tabu Search described above proved quite effective, especially when dealing with problems with a small number of commodities, its performance is not totally satisfactory when one must solve network design instances with a large number of commodities, as one often encounters in practical applications. The main reason for this is that the moves considered in the extreme-point neighborhood only consider flow modifications of a single commodity at the time. This turns out to be too myopic for instances with several commodities.

One way to address this is to revisit approaches based on the exploration of the search space of design (i.e., binary) decision variables. As we have mentioned earlier, simple neighborhood structures for this search space, such as add-drop and swap, are clearly ineffective. One must thus look for new neighborhood structures that allow for a thorough and efficient search of the space of design variables. In this quest, one must first note that in order to significantly modify a solution in a network design problem, one must be able to open and close simultaneously sequences of arcs that make up subpaths. Furthermore, given a specific complete solution (i.e., including the values of the continuous variables), possible flow movements can only take place along cycles in the *residual graph* with respect to this solution.

The cycle-based neighborhood relies on the identification of cycles of given capacities in the residual graph. This is achieved by considering sets of candidate

arcs as starting points for creating cycles; then, a labeling heuristic is used to identify low-cost cycles containing each of the candidate arcs. For each solution, once the cycle has been identified, the flow pattern is adjusted by solving exactly the associated MCMNF problem defined by (4.15)–(4.18), where $\bar{\mathbf{y}}$ is the design solution being evaluated. One of the most advantageous features of this new neighborhood structure is that it allows significant modifications of the current solution at each iteration, involving simultaneous flow changes for several commodities on several arcs.

To speed up the exploration of the cycle neighborhood, the flows of all commodities are aggregated in residual graphs. Because of that, the MCMNF problem may turn out to be infeasible, which requires the use of a suitable *restoration* procedure to retrieve a feasible solution.

The intrinsic power of the cycle neighborhood allows for the use of a fairly simple global search strategy. Hence, search intensification, implemented through flow modifications of single commodities, is invoked when very good solutions are obtained. With respect to search termination, very simple criteria, such as the total number of iterations or the global CPU time elapsed, can be used.

A comprehensive computational experimentation on the **C** instances of the Gendron-Crainic benchmark has shown that Tabu Search based on cycle neighborhoods is much more effective than the pivot-based approach described previously.

## *4.2 Other Neighborhood-Based Metaheuristics*

In this section, we describe rapidly four families of metaheuristics, which, to the best of our knowledge, have not been yet applied directly to the solution of basic network design problems in the context of transportation and logistics. The main reason for presenting these families is that most of them are used as part of hybrid methods or matheuristics presented later in the chapter. In some cases, these methods have been applied to more complex variants of the problems that we discuss, as will be mentioned in the bibliographical notes of Sect. 8.

It is important to note that all these methods could be used, by themselves, to tackle the fixed-charge transportation or the multicommodity capacitated fixed-charge network design problems using some of the neighborhoods defined earlier.

### 4.2.1 Simulated Annealing

Simulated Annealing (SA) is one of the oldest and simplest metaheuristics. According to an analogy with the cooling of material in a heat bath, solutions to an optimization problem correspond to configurations of particles and the value of the objective function to the energy of the system for a given configuration. The trajectory followed by an SA procedure can be interpreted as a *controlled random walk* in the search space: at each step, a random solution is generated in the

neighborhood of the current solution; if this new solution leads to an improved solution, the new solution is accepted and becomes the current one; if the tentative move deteriorates the objective, it is performed subject to a probabilistic acceptance criterion, which depends on the magnitude of the deterioration and a search control parameter, called the *temperature*. This temperature is slowly decreased with time, making non-improving moves more and more difficult to accept. Interestingly, under suitable assumptions on the number of temperature levels and the number of iterations per level, Markov chain theory can be used to show that SA should converge asymptotically to the global optimum of the problem at hand. In practice, however, these assumptions cannot be fulfilled within reasonable computation times. More relevant is the fact that, if one records the best solution observed during the random walk in the search space, a high-quality solution can often be identified within a reasonable computational effort.

Deterministic versions were also proposed under the names of *deterministic annealing* and *threshold acceptance*.These methods are similar in the sense that they are neighborhood search methods in which deterioration of the objective up to a *threshold* is accepted, the threshold decreasing as the algorithm progresses.

### 4.2.2 Iterated Local Search

Iterated Local Search (ILS) is a rather straightforward metaheuristic that, in its simplest form, combines basic Local Search with the concept of *perturbation*. Thus, ILS escapes from local optima by applying random perturbations to the current local optimum $s^\star$ to produce an intermediate solution $s'$, to which Local Search is again applied. This leads to another local optimum solution $s'^\star$, which can be selected according to some *acceptance criterion*; if the solution $s'^\star$ does not satisfy the criterion, the search returns to $s^\star$, from which a new perturbed solution $s''$ is created. In many implementations, the acceptance criterion is very simple: $s'^\star$ is accepted as the new current solution only if its objective function value is better than the value of $s^\star$. Obviously, more sophisticated acceptance criteria may be used, and perturbation schemes may be devised to account for the past history of the search.

### 4.2.3 Greedy Randomized Adaptive Search Procedure

Greedy Randomized Adaptive Search Procedure (GRASP) is a fairly straightforward iterative metaheuristic that combines the simplicity of greedy solution heuristics with the power of randomized algorithms to tackle difficult combinatorial problems. Typically, each iteration of a GRASP metaheuristic is made up of two phases: a construction phase and a Local Search phase. In the former phase, a solution to the problem at hand is constructed by selecting one element at the time. The selection process follows greedy principles, but in randomized fashion: instead of selecting

the most attractive element, a *restricted candidate list* (RCL) containing a subset of the most attractive elements is maintained and an element of RCL is chosen at random to be included in the solution. This procedure is repeated until a full solution is constructed. In the second phase, a suitable Local Search procedure is applied to the solution just constructed. It is important to note that the procedure can be applied a large number of times, since the choices made in the construction phase, being randomized, will lead to different solutions. One can thus consider a large number of possible solutions to the problem at hand. GRASP typically stops after performing a given number of iterations.

### 4.2.4 Variable Neighborhood Search

Variable Neighborhood Search (VNS) is a neighborhood-based metaheuristic that strives to perform a more effective exploration of the search space by exploiting several neighborhoods, but *not simultaneously*, i.e, different neighborhoods are considered in sequence. In its simpler version, which is called *Variable Neighborhood Descent (VND)*, neighborhoods are considered individually until one runs into a local optimum; when this happens one switches to the next neighborhood in the sequence. Several strategies can be applied with respect to the exploration of the neighborhoods. One may, e.g., restart the search using the first neighborhood or consider neighborhoods cyclically. The search terminates when none of the considered neighborhoods can lead to a better solution; this corresponds to having found a local optimum for each of these neighborhood structures. In general, if several neighborhoods are examined, the final solution should be an excellent one.

Other forms of VNS consider more intricate search mechanisms. One important mechanism is the use of a *shaking function* to generate points at random in some neighborhood before starting the exploration. Many of the advanced versions of VNS combine deterministic and stochastic changes of neighborhoods.

## 5 Population-Based Metaheuristics

As was already mentioned, population-based metaheuristics explore a suitably defined search space for the problem at hand by evolving a population of solutions through the applications of combination mechanisms and other procedures, which depend on the specific method considered. There is a wide range of population-based metaheuristics, but, in this chapter, we will focus on three families that have proved useful for solving network design problems: (1) Genetic and Evolutionary Algorithms, (2) Path Relinking procedures, and (3) Scatter Search.

## 5.1 Genetic Algorithms/Evolutionary Algorithms

Genetic Algorithms (GAs) are probably the first metaheuristic, going back to the mid-1970s, more than 10 years before the term metaheuristic was coined by Glover (1986). The basic GA is based on an analogy with the Darwinian evolution principles, the fundamental idea being to replicate the evolution of a population of *individuals*, which represent solutions to the problem at hand. Solutions and, indirectly, the search space are encoded as *chromosomes*. In early implementations of GA, chromosomes were simply bit strings, but natural encoding has become the norm in later implementations. The selection of individuals is performed according to their *fitness*, high fitness values corresponding to highly-desirable individuals. The fitness measure may be the objective function value associated with the individual (solution), or more complex measures accounting, e.g., for the distance from an "ideal" or the average measure for the population.

The basic GA mechanisms revolve around three key operators: *selection*, which identifies which individuals should be selected as *parents* for *reproduction*, i.e., to be used for creating new individuals; *crossover*, which creates one or two *offsprings* from the genetic material (i.e., features) of a pair of parents; and *mutation*, which randomly modifies the value of some *gene* (basic element of a chromosome). Selection is based on the fitness of individuals, often with stochastic elements, such as in *roulette-wheel selection*; the rationale is that one hopes that better offspring will be obtained from the best parents. Basic crossover operators, based on the bit-string representation of chromosomes, were originally proposed, but, over time, more sophisticated crossovers were designed for specific classes of problems.

Typically, a GA runs for a number of *generations* (iterations). In each generation, some individuals are selected to reproduce; then, new individuals are created from these by the application of the crossover operator; and mutation is applied to their offspring. The resulting new individuals are then added to the population, usually replacing less fit ones or all of them. It should be noted that some GAs simply create and introduce offspring in the population without considering generations.

A critical element in the successful application of GA is the need to maintain *diversity* inside the population: if individuals in the population become too similar, the method loses its ability to properly explore the search space. This realization has led to the introduction of the concept of *biased fitness*, which combines diversity considerations in the fitness function.

Over time, it has also become obvious that pure GA methods, implementing the traditional operators only, are usually not powerful enough to address hard combinatorial problems. This has led to the development of hybrid GAs that use some form of neighborhood search to improve offspring. In this case, one often talks of an *education* operator.

Evolutionary Algorithms (EAs) can be seen as an extension and generalization of GAs. Mutation plays a much more important role than recombination, mutation operators being often much more sophisticated than in GAs. Moreover, in many EAs, the representation of individuals includes strategy parameters in addition to the solution vector.

### 5.1.1 A Genetic Algorithm for the Fixed-Charge Transportation Problem

Several GAs have been proposed for tackling the FCTP. We present a fairly recent implementation.

This GA explores the set of basic solutions of the standard formulation of the transportation problem obtained by opening all arcs of the bipartite graph, as in Sects. 3.2.2 and 4.1.1. However, the representation of these solutions is quite different from what we have seen in these subsections. The GA uses a so-called *priority-based encoding* of solutions. In this scheme, each solution (i.e., individual or chromosome) is represented using a vector of length $(m + n)$, where $m$ is the number of sources and $n$ is the number of of destinations. Each chromosome corresponds to a permutation of the integers between 1 and $(m + n)$. The first $m$ genes (entries) of the chromosome are associated with sources and the $n$ last ones with destinations. The value of a specific gene indicates the priority given to a source or a destination when decoding the chromosome, with $(m + n)$ being the highest priority and 1 the lowest. The decoding of a chromosome is performed by considering the residual supply (resp. demand) in source (resp. destination) nodes and entries in the cost matrix for the problem, in a process reminescent of the greedy procedure of Sect. 3.1. The main difference, however, is that arcs on which flows will be added are selected on the basis of the priority-based representation. Considering the properties of the transportation problem, the decoding of a chromosome always result in a feasible solution whose fitness can be evaluated easily.

Because of its specific nature, the priority-based encoding requires the use of specialized crossover and mutation operators. Two new crossover operators are proposed: the *order of priority exchange* crossover (OPEX) and the *priority exchange* crossover (PEX). Both of these crossovers return two priority-based encoded offsprings from two parents in the same encoding. A specialized mutation operator, based on the OPEX crossover applied to two segments of a chromosome, is also used. The method uses a mixed selection strategy for population management. Computational experiments showed that the proposed GA would systematically outperform GAs with a spanning-tree based representation of chromosomes. Furthermore, the method can easily be adapted to tackle quadratic flow transportation costs.

### 5.1.2 A Genetic Algorithm for the Multicommodity Capacitated Fixed-Charge Network Design Problem

Let us now describe a fairly straighforward application of GA mechanisms to solve the MCFND. The search space for this implementation is the space of feasible binary design vectors. The fitness of any $\bar{\mathbf{y}}$ vector is given by the sum of the fixed costs of the arcs open in $\bar{\mathbf{y}}$ and the optimal value of the MCMNF problem associated to $\bar{\mathbf{y}}$ and defined by (4.15)–(4.18). The best individuals are thus those with a low fitness.

The method proceeds on a generation by generation basis. In each generation, a number of pairs of parents are selected for reproduction on the basis of their rank (w.r.t. to fitness) within the population, in accordance with roulette-wheel selection schemes. This selection scheme avoids a premature convergence of the population because of the dominance of *super-individuals* who would be repeatedly selected.

Two crossover operators are examined. Both of them yield a pair of offsprings from each pair of parents. The first one is the *uniform crossover*, which is based on the application of a randomly generated binary uniform mask $m$ of the same length as the chromosomes. Each entry in $m$ takes the value 0 or 1 with probability 0.5. When $m(i) = 1$, the $i$-th gene of the first offspring $O_1$ is copied from the $i$-th gene of the first parent $P_1$ and the $i$-th gene of the second offspring $O_2$ is copied from the $i$-th gene of the second parent $P_2$. When $m(i) = 0$, the roles of $P_1$ and $P_2$ are reversed.

A second crossover, called the *frequency crossover*, attempts to incorporate some of the knowledge gathered from the characteristics of good solutions to the problem at hand. Let $\phi_i$ be the frequency of apparition of arc $i$ in good solutions. A randomly generated binary mask $m$ in which the probability that $m(i) = 1$ is equal to $\phi_i$ is defined. The frequency crossover then works as follows: when $m(i) = 1$ and the $i$-th gene of the first parent $P_1$ is also equal to 1, then the $i$-th gene of the first offspring $O_1$ is set equal to 1 and the $i$-th gene of the second offspring $O_2$ is copied from the $i$-th gene of the second parent $P_2$; otherwise (i.e., when $m(i) = 0$ or when the $i$-th gene of the first parent $P_1$ is equal to 0), then the $i$-th gene of the first offspring $O_1$ is copied from the $i$-th gene of the second parent $P_2$ and the $i$-th gene of the second offspring $O_2$ is copied from the $i$-th gene of the first parent $P_1$, as in the uniform crossover when $m(i) = 0$. Feasible offsprings are added to the population. When the population reaches a given threshold, a number of least fit individuals are eliminated according to a $(\lambda + \mu)$ population management strategy.

The GA may be stopped after a set number of generations, of new offsprings, or when the population diversity becomes too low. This GA is also part of a hybrid parallel search method, which will be described in Sect. 7, and it proved quite successful in this context.

## 5.2 Path Relinking

Path Relinking was specifically developed to intensify and diversify the exploration of promising portions of the search space of combinatorial optimization problems. As such, it is not meant to be used as a stand-alone method, but rather to complement other solution procedures, which are often neighborhood-based metaheuristics.

The basic idea is to first construct, using some other method, an initial *reference set* of solutions (this is the initial population for this method). This reference set should contain, as much as possible, excellent solutions (they are called *elite* solutions), but one should also strive to maintain some level of diversity of solutions in the reference set. The main step of the method consists of choosing in the

reference set two solutions, an *initial solution* and a *guiding solution*, and then following a path in the search space from the initial solution towards the guiding one, by gradually introducing in the initial solution features of the guiding solution. The rationale behind this procedure is that paths linking good solutions have an excellent chance of containing even better solutions. When improving solutions are found on a path, they can be added to the reference set. The exploration of the path between the initial and the guiding solutions can be stopped before reaching the guiding solution. At that time, new initial and guiding solutions are selected and the process repeated. Path Relinking can stop after a set CPU time or a given number of selections of initial and guiding solutions.

### 5.2.1 Path Relinking for the Multicommodity Capacitated Fixed-Charge Network Design Problem

Path Relinking was applied to the MCFND problem in the context of an extensive computational study. The basic metaheuristic applied was the Tabu Search that uses cycle-based neighborhoods. Different strategies were considered for collecting elite solutions and creating the reference set:

- (S1): Best solutions
- (S2): Best local minima found during the search
- (S3): Local minima that improve those already in the reference set
- (S4): Solutions far from those previously chosen and better than the worst one
- (S5): Best solutions, then extend with solutions far from those already chosen
- (S6): Best solutions, then extend with solutions close to those already chosen

Several criteria were also considered for the selection of the initial and the guiding solutions:

- (C1): Best and worst
- (C2): Best and the second best
- (C3): Best and the solution with the maximum Hamming distance from best
- (C4): Randomly
- (C5): The most distant solutions
- (C6): Worst and best

Trajectories between initial and guiding solutions are explored using a variant of the cycle-based neighborhood in which the flows of a single commodity are modified each time. This procedure allows to progressively introduce in the current solution arcs that are present in the guiding solution, but not not in the current one, and to remove arcs that do not belong to the guiding solution. The best solution obtained during a relinking process is added to the reference set, if it improves the best overall solution.

Computational experiments on both the **C** and the **R** instances of the Gendron-Crainic benchmark highlighted a number of important conclusions. First, it is not necessary to use a large reference set; in the experiments, using a reference set of

size 6 yielded the better performance. Second, diversity and long relinking paths provide the most effective exploration and the best solutions in the end; hence, the combination of strategy (S3) and criterion (C5) led to the best results overall. Third, when it is properly used and calibrated, Path Relinking is an effective approach for tackling multicommodity capacitated fixed-charge network design problem.

## *5.3 Scatter Search*

Scatter Search is another technique that has been proposed to combine good known solutions for a problem in the hope of finding even better ones. It shares many concepts with Path Relinking, such as the use of a reference set. The basic mechanism is to perform a weighted linear combination of two or more vectors that represent solutions extracted from the reference set. As such, it is therefore naturally suited to tackle continuous optimization problems. It can, however, be applied to problems with integer decision variables, such as network design problems.

### 5.3.1 Scatter Search for the Multicommodity Capacitated Fixed-Charge Network Design Problem

The search space for the application of Scatter Search to the MCFND encompasses the binary design variables. Since we are dealing with integer variables, the basic Scatter Search mechanism is adapted by resorting to sophisticated rounding procedures to fix some of the binary variables and letting others be free.

At the start of the algorithm, the cycle-based Tabu Search is applied to create an initial population of solutions, from which the reference set is initialized with with improving local optima, i.e., local optima better than solutions already in the set.

At each iteration, a *candidate set* $CS$ of $L$ solutions is created by including the best solution in the reference set, the solution that is the farthest away from it in Hamming distance, and a number of randomly selected elements of the reference set. The binary vectors for these $L$ solutions are then combined using weights $\omega_l, \forall\, l \in CS$, to compute a *desirability factor* $m_{ij}$ for each arc $(i, j) \in \mathscr{A}$. The desirability factors are then compared for each arc to two thresholds $t_c$ and $t_o$, with $0 < t_c < t_o < 1$:

- When $0 \leq m_{ij} < t_c$, we close arc $(i, j)$ in the new solution;
- When $t_o < m_{ij} \leq 1$, we open arc $(i, j)$ in the new solution;
- When $t_c \leq m_{ij} \leq t_o$, we leave arc $(i, j)$ undecided in the new solution.

Four different weighting schemes were tested for combining vectors of the candidate set. These are:

- Voting (V): $\omega_l = 1, \forall\, l \in CS$;
- Cost (C): $\omega_l = 1/$(cost difference between solution $l$ and the best solution);

- Distance (H): $\omega_l = 1/$(Hamming distance between sol. $l$ and the best solution);
- Frequency (F): $\omega_l =$ frequency of arc $(i, j)$ in the best solutions.

The incomplete solution obtained from the combination procedure is then further processed. First, a special minimum cost multicommodity network flow problem, similar to (4.15)–(4.18), is solved using CPLEX. In this MCMNF problem, arcs that have been closed by the rounding scheme have their capacity set to 0, while the variable cost of undecided arcs is set to $f_{ij}/u_{ij} + c_{ij}$. A feasible solution for the MCFND is extracted from the MCMNF optimal solution by opening all arcs with flow; this solution is then improved by applying the cycle-based Tabu Search.

Preliminary testing on a small set of instances permitted to identify the best values for the thresholds: 0.4 for $t_c$ and 0.6 for $t_o$. Extensive computational experiments on the **C** instances of the Gendron-Crainic benchmark tested different values for the size $L$ of the candidate set and the size of the reference set, as well as the various weighting schemes. These experiments led to the following conclusions. First, there should be at least 20 elements in the reference set. Second, results with only two elements in the candidate set are clearly inferior; results with $L = 3, 4,$ or 5 are globally comparable; in practice, it seems that $L = 3$ or 4 is the best choice. Third, while all combinations of $L$ and weighting schemes can produce on some instance results that are better than those of the Path Relinking implementation of Sect. 5.2.1, on average, none of the combinations does as well. Fourth, the Frequency weighting scheme is clearly inferior to the other three. Overall, the observed performance seems to indicate that the full potential of Scatter Search has not been exploited in the proposed method.

### 5.3.2 An Improved Scatter Search-Evolutionary Algorithm for the Multicommodity Capacitated Fixed-Charge Network Design Problem

We now consider a more involved method involving Scatter Search to tackle MCFND problems. This method can be decomposed into three major phases: an Initialization phase, a Scatter Search phase, and an Education phase based on Iterated Local Search (ILS). While the Initialization phase is executed only once at the beginning of the algorithm to produce an initial population of solutions stored in reference set $R$, the two other phases are performed iteratively to improve solutions in set $R$. The termination criterion of the algorithm is the elapsed CPU time.

The Initialization phase seeks to produce a population of solutions that are of good quality, but also diverse. The solutions considered at this step include both the binary design variables and the flow variables. These initial solutions are constructed by gradually constructing shortest paths for all commodities between their origin and their destination; the sequence in which commodities are routed is randomized and single or multiple paths may be constructed for each commodity. $\lambda$ solutions are created, but only $\mu$ with $\mu < \lambda$ are retained. In fact, the first $\mu$ solutions created are first assigned to $R$ and each one of the remaining $(\lambda - \mu)$ is considered for replacing

the worst solution in $R$, if it cost is lower than that the cost of the best solution in $R$ or if it dominates some solution in $R$ both in terms of solution quality and solution diversity (measured in terms of Hamming distance to the best solution).

It should be noted that the search that takes place in the following phases is performed in the space of the binary design variables. The values of the flow variables for a given design are obtained by solving the associated MCMNF problem, as in several of the other solution methods that we have seen for the MCFND.

In the Scatter Search phase, solution recombination takes place. In each iteration, a total of $2\mu$ offsprings are created, each one being determined by combining features from a *candidate set* $CS$ made up of $\kappa$ solutions. The selection procedure for choosing the solutions that make up the candidate set is probabilistic, but it does not involve the cost of solutions directly. Instead, as the method proceeds, information on the performance of solutions with respect to the Education phase is recorded, for each, in a measure which is called its *solvency ratio* (SR). For any given solution $s$, SR is the ratio of the number of times that an offspring of solution $s$ has been included in $R$ (after education) to the number of times that solution $s$ has been selected to produce an offspring. It is thus a measure of the effectiveness of solution $s$ to produce useful offsprings for future generations. The recombination process itself is based, for each arc $(i, j)$, on a weighted sum of the design variables of the solutions of $CS$ for this arc. The weight given to each depends upon its objective value and a correction factor to lessen the importance of solutions that have appeared often in $CS$. This computation allows to determine a *preferred status* of open or closed for each arc in the offspring. The corresponding binary vector is then a tentative design solution, for which one can solve the associated MCMNF problem with a linear programming solver. If this problem is feasible, the offspring can proceed; if it is infeasible, the design must be repaired using a process similar to the one performed in the Initialization phase. At the end of the Scatter Search phase, the best $\mu$ offsprings, in terms of solution cost, are retained to proceed to the next phase.

In the Education phase, the $\mu$ best offsprings coming from the Scatter Search phase go through a process aimed at improving their quality. Each offspring is educated individually by going through an ILS procedure. This ILS procedure has two components: a neighborhood search that can be seen as an extension of the cycle-based Tabu Search of Sect. 4.1.2, and a perturbation strategy, called *Ejection Cycles*, which partially modifies the current solution using information stored in a long-term memory of the search.

This method was applied to the **C** instances of the Gendron-Crainic benchmark. The results obtained show that this method is very competitive with the best existing ones (including the methods described in the next section) and could identify new best solutions for some instances.

## 6 Matheuristics

Matheuristics is a broad term that covers all solution approaches that combine exact solution procedures with metaheuristic methods. As such, there is no set recipe for deriving matheuristics; each matheuristic is a unique method and should be described as such for the time being. Perhaps, in the future, someone will propose a taxonomy of matheuristics and provide a more insightful description of these methods. In this chapter, we present four interesting matheuristics that have been proposed for tackling network design problems.

### *6.1 A Local Branching Matheuristic for the Multicommodity Capacitated Fixed-Charge Network Design Problem*

*Local Branching* (LB) is a matheuristic that was developed in the early 2000's to leverage the power of mixed integer programming (MIP) solvers. It is particularly aimed at combinatorial optimization problems that can be modeled with binary decision variables. The central idea of LB is to explore partially the space of feasible binary vectors using depth-first tree search. At each iteration, the search is limited to considering a fairly small neighborhood of a *reference solution* by the addition of so-called *local branching constraints*, which require solutions to differ from the reference solution by no more than $k$ values. This defines what is called a *k-Opt* neighborhood. The reference solution is updated whenever an improving incumbent is found. In theory, if the underlying tree search was performed completely, LB would be an exact method, but the addition of CPU time limits and bounds on the depth of some branches of the search tree turns it into an approximate procedure.

Network design problems with their binary design decision variables are natural candidates for the application of LB, which was thus applied to the MCFND. The MIP formulation used in the application is the arc-based formulation of Chap. 2 to which are added the *strong linking constraints* described in this chapter, as well as the local branching constraints. An important feature of the method as implemented is that the MIP considered at each node of the search tree is not solved to optimality; as soon as a feasible solution that improves upon the value of the reference solution is found, the values of the design variables are fixed and optimal values for the flow variables are derived from the resulting MCMNF. This procedure guarantees that the proper objective function value is determined for the corresponding design.

Computational experiments on the **C** instances of the Gendron-Crainic benchmark showed that an LB-based matheuristic could lead to improved solutions for several instances in reasonable CPU times.

## 6.2 A Matheuristic Combining Exact and Heuristic Approaches for the Multicommodity Capacitated Fixed-Charge Network Design Problem

We now describe a method that was developed with the objective of producing provably high-quality solutions quickly for the MCFND. The method relies on a neighborhood search procedure that explores the space of binary design variables using a MIP solver, in a fashion reminescent of the method presented in the previous subsection. In each iteration, a subset of design variables of the arc-based formulation of the MCFND is thus chosen and "freed", while the other binary variables are held fixed, and the resulting MIP is solved. Several strategies for choosing which arcs should be freed are used; these involve identifying arcs that are used in some commodity paths of the current solution, in paths derived from solving some linear relaxations, in paths that were observed in improving solutions, and so on. Proper choice of the restricted IP and proper seeding of the initial solution used to solve it should result in most cases in improved solutions and thus better upper bounds.

A lower bound on the optimal value of the problem is also computed at each iteration. It is provided by the value of the LP relaxation of the path-based formulation. However, since this bound is known to be weak, valid inequalities are added to this formulation. These include strong linking inequalities between the design and the flow variables, as well as lifted cover inequalities found while solving the small MIP at each iteration.

The proposed approach was tested on a comprehensive set of instances. In particular, it was used to tackle the 37 most difficult **C** instances of the Gendron-Crainic benchmark. On these instances, it clearly outperformed the methods presented in Sects. 4.1.2 and 5.2, both in terms of solution quality and computing times. It was also able to find in a few minutes solutions that were on average only 2.37% more expensive than to those obtained by CPLEX in several hours of computation. When tested on larger instances with 500 nodes, 2000–3000 arcs, and 50–200 commodities, it produced in 15 min of CPU time solutions that were on average more than 20% better than those obtained by CPLEX after 12 h of computation.

It is important to note that this solution approach can tackle not only the MCFND basic formulation presented in Chap. 2, but also its variant in which commodities must be routed on a single path between their origin and their destination. This is partly due to the fact that the mathematical formulations used are not exactly the ones presented in Chap. 2: instead of using variables $x_{ij}^k$ to represent the flow of commodity $k$ on arc $(i, j)$, these variables denote the *fraction of the demand* of commodity $k$ that uses arc $(i, j)$. Single-path constraints for commodities are easily enforced by requiring these $x_{ij}^k$ variables to be binary.

## 6.3 A Hybrid Simulated Annealing-Column Generation Matheuristic for the Multicommodity Capacitated Fixed-Charge Network Design Problem

This method is a fairly straightforward hybridization of Simulated Annealing to determine the values of the binary design variable and Column Generation to solve the path-based formulation of the MCMNF problem for a given vector of binary variables $\bar{\mathbf{y}}$. Thus, the SA heuristic explores the search space of binary design vectors using moves in the add-drop neighborhood. Several strategies are considered for choosing the arcs to be closed, sometimes complemented by the opening of arcs along some paths when the current solution becomes infeasible. The objective function associated with any solution is the sum of the fixed costs of open design arcs plus the objective value of the corresponding MCMNF problem. Solutions are accepted (and thus moves performed) according to standard SA acceptance rules. The temperature is adjusted after a fixed number of add/drop cycles; it is lowered according to a linear function. In this method, the methods stops after a examining pre-determined number of temperature settings without improvement.

Extensive analysis was performed to tune the various parameters of the method. With the best parameter values, the method was shown to perform quite well. On the 43 **C** instances of the Gendron-Crainic benchmark, it produced better results than the Local Branching algorithm of Sect. 6.1 and CPLEX with a time limit of 600 s. These conclusions were shown to be statistically significant.

## 6.4 A Cutting-Plane Based Matheuristic for the Multicommodity Capacitated Fixed-Charge Network Design Problem

This matheuristic is based on a new neighborhood structure which relies on cutting planes for its definition. The fashion in which the neighborhood is defined is procedural. Given an incumbent (current) solution, an arc open in this solution is selected to be closed. Several criteria were considered for selecting that arc, but in the end two were retained: (1) the arc with the maximum combined unit cost (including fixed and variable costs), given its current flow; (2) the arc with the highest fixed cost for residual capacity (i.e., the product of the arc fixed cost by the ratio of unused to total capacity). These two criteria point out to open arcs whose usage does not seem to be very efficient. Then, the continuous (LP) relaxation of the MCFND is solved with two additional constraints: (1) a constraint closing the arc selected in the previous step, and (2) a constraint requiring the number of open arcs (other than the one selected to be closed) to remain the same as in the incumbent. If this LP is infeasible, a new open arc is selected to be closed and the procedure is repeated. Once a feasible solution to the LP is found, families of valid inequalities

are added to this LP. Three families are considered: (1) strong linking constraints (presented in Chap. 2) between individual commodity flows and design variables; (2) flow cover inequalities, and (3) flow pack inequalities. It should be noted that strong linking constraints are added to the LP as long as they increase its objective value. As for the flow cover and flow packing inequalities, they are identified by applying a separation algorithm presented in Chouman et al. (2009). These valid inequalities and the constraint on the number of open arcs define a so-called *strong LP*. A new MIP sub-model is then defined from the strong LP by adding to it three constraints: (1) one that forces to open arcs that carry flow in both the strong LP solution and the incumbent; (2) a constraint that closes arcs that do not carry flow in both the strong LP solution and the incumbent; and (3) the constraint that forces to close the arc selected at the beginning of the procedure. The resulting MIP is solved by Local Branching, as in the method described in Sect. 6.1. The solution obtained is the desired neighbor to the current solution.

The proposed neighborhood structure was used within a Tabu Search heuristic. The initial solution for this Tabu Search is a feasible one, which is obtained by solving the LP relaxation of the problem, adding valid inequalities as in the procedure that defines the neighborhood structure, and rounding up the value of binary design variables in this strong LP. Two tabu lists are used. The first one records the list of recently closed arcs; these arcs are forced to remain closed for a number of iterations to prevent cycling. The second tabu list is used whenever the new solution produced by the neighborhood structure is infeasible or worse than the incumbent solution. In that case, several moves are attempted from the incumbent until an improving one is found or a pre-defined number of non-improving neighbors is reached; the tabu list is used to record the arcs closed in the unsuccessful attempts. The procedure stops after a pre-defined CPU time or a set number of iterations without improvement of the incumbent.

The values of the parameters of the proposed tabu search heuristic were carefully tuned using a statistical design of the experiments. Computational results on 37 of the 43 **C** instances of the Gendron-Crainic benchmark (the six easiest problems were left out) showed the effectiveness of the proposed methods: it clearly outperformed the oldest methods and did as well or better than the most up-to-date.

## 7 Parallel Metaheuristics

Parallel/distributed/concurrent computing means that several processes work simultaneously on several processors addressing a given problem instance and aiming to identify the best or a good feasible solution for that instance. Parallel metaheuristics and matheuristics aim for two major goals. The first, common to all parallel-computing developments, is to solve larger problem instances, faster. The second is proper to heuristics and it concerns the robustness of the search, i.e., its capability to offer a consistently high level of performance over a wide variety of problem settings and instance characteristics. Parallel metaheuristics built according to cooperative

multi-search strategies have thus proved to be much more robust than sequential versions, offering higher quality solutions, and requiring less extensive, and expensive, parameter-calibration efforts. It is worth noticing that multi-search methods, particularly when based on cooperation, generally display a behavior different from those of the sequential methods involved, offering enhanced performance compared to sequential methods and other parallelization strategies. They are thus acknowledged as proper metaheuristics.

The objective of this section is to present a brief unified overview of the main parallel metaheuristic concepts and strategies. Note that these are of a general nature with respect to the metaheuristic type and combinatorial optimization problems. We will identify network design applications, of course, and will signal relevant particularities for neighborhood- and population-based meta- and matheuristics.

Parallelism follows from a decomposition of the algorithmic work required to address the problem, and the distribution of the resulting tasks to available processors. The decomposition may concern the algorithm, the search space, or the problem structure. Parallel strategies may then be classified according to, on the one hand, what is decomposed and how it is done, and, on the other hand, how the global problem-solving search is controlled, how information is exchanged among tasks and how, eventually, new information is created (the diversity of the searches involved may also be used, but we skip it in the following for chapter-length reasons).

*Functional parallelism* and *search-space separation* make up the first dimension. Within each decomposition strategy, strategies are then described in terms of *Search Control Cardinality*, specifying whether the global search is controlled by a single (*1-control*—1C) or several (*p-control*—pC*)* processes, which may collaborate or not, as well as of *Search Control and Communications*, addressing how information is exchanged and used to control or guide the search. *Synchronous* and *asynchronous* communications are found in parallel computing. All processes stop, at moments exogenously determined, in the former case, to engage in some form of communication and information exchange. In the latter case, each process is in charge of its own search and of establishing communications and exchanges with other processes. Four categories are defined to reflect the quantity and quality of the information exchanged and shared, as well as the additional knowledge derived from these exchanges (if any); *Rigid* and *Knowledge* synchronization, and *Collegial (C)* and *Knowledge Collegial (KC)* asynchronous strategies. These concepts and their applications to network design is further described in the next subsections.

## *7.1 Functional Parallel Strategies*

*Functional* or *low-level* parallelism decomposes computing-intensive parts of the algorithm into a number of tasks, which work on the same data or on a dedicated part, and run in parallel. Such strategies thus aim to accelerate the search, without modifying the algorithmic logic, the search space, or the behavior of the sequential

metaheuristic. Most low-level parallel strategies belong to the 1C/RS class and are usually implemented according to the classical *master-slave* parallel programming model. A "master" program initiates the exploration from a single solution or population. It executes the sequential metaheuristic (1-control), separating and dispatching computation-intensive tasks, which often corresponds for metaheuristics to the execution of the innermost loop iterations, e.g., evaluating neighbors or individuals, or having ants forage concurrently. Slaves perform the tasks in parallel, and return the results to the master which, once all the results are in, resumes the normal logic of the sequential metaheuristic. The master has complete control on the algorithm execution; it decides the work allocation for all other processors and initiates communications. No communications take place among slave programs.

Functional parallelism is often a low-level component of hierarchical parallelization strategies and may be extremely interesting when the problem requires a significant part of the computing effort to be spent in inner-loop algorithmic components. This is often the case for network design, in particular when search spaces are based on the design decision variables. The evaluation of neighbors and individuals often requires in this case to find the optimal, or at least a very good solution to the multicommodity capacitated network flow subproblem, which may be quite computationally intensive. The parallel evaluation of several neighbors, of the fitness and diversity measures of individuals, or of scatter-search operators on several combinations of solutions in the reference set could be of significant help in those circumstances. Note that similar observations may be made for pivot-based moves. Note also that one could use the *graphical processing units* (*GPU*), ubiquitous within most computers, to further parallelize pivot operations, either as moves, or as part of customized linear-programming solvers. This is an interesting research area.

## *7.2 Search-Space Separation: Domain-Decomposition Strategies*

The general idea of the second major class of parallel strategies is to decompose the search space and to address the problem on each resulting component using a particular solution method. Two main classes of such strategies may be defined: *domain decomposition* and *multi search*. The former explicitly separates the space yielding a number of subproblems to be addressed simultaneously, their solutions being then combined into a complete solution to the original problem, while the latter performs the separation implicitly, through the concurrent explorations of the complete space by several methods, named *solvers* in the following, which may exchange information or not. Multi-search strategies, particularly those based on cooperation principles, are at the core of most successful developments in parallel metaheuristics, particularly for complex, multi-attribute combinatorial problem, as those found in network design and applications. Note that search-space decompo-

sition methods induce different search behaviors and yield different solutions when compared to the corresponding sequential metaheuristics involved.

The basic idea of *domain decomposition* is intuitively simple and appealing: separate the search space into smaller subspaces, address the resulting subproblems by applying the sequential metaheuristic on each subspace, collect the respective partial solutions, and reconstruct an entire solution out of the partial ones. The subspaces may be disjoint, their union yielding the full space, or a certain amount of subspace overlap may be allowed, either explicitly, or implicitly by allowing the search within a given subspace to reach out to some part of one or several other subspaces through, e.g., neighborhood moves or individual crossovers. The separation may be obtained by identifying a subset of variables (and corresponding constraints, eventually) and discarding or fixing the other variables and constraints, the goal being to obtain smaller, easier to address subproblems. For network design, separation could thus be defined along groups of arcs, nodes (commodity origins or destinations), commodities, paths of potential design arcs, etc., based on attributes of the corresponding elements (e.g., proximity of nodes or arcs fixed cost over capacity ratio) or solutions at previous iterations. Discarding does not appear appropriate when considering the commodities and arcs of multicommodity capacitated network design problems, however. Separation by variable fixing (and projection of the corresponding constraints) appears more flexible as one still works on smaller subproblems, but considering the complete vector of decision variables, some of which are fixed. This is also a more general approach and it is part of advanced cooperative search methods described later.

Notice that strict partitioning restricts the solvers to their subspaces, resulting in part of the search space being unreachable and the loss of exploration quality. Covers partially address this issue. Yet, to guarantee that all potential solutions are reachable, one must make overlapping cover the entire search space, which negates the benefits of decomposition. The "change the separation and start again" idea is at the core of the 1C/RS and pC/KS strategies proposed to avoid these drawbacks, where the separation is modified periodically, and the search is restarted using the new decomposition. The master (in the former case, or the collaborating processes, in the latter) applies the decomposition, reconstructs complete solutions, modifies the separation, and determines when stopping conditions are met.

An application of this strategy to the MCFND may partition the search space of full feasible solutions based on commodities and variable fixing. Starting from a given (initial) solution, one forms partitions of commodities sharing at least one design arc (a graph-partitioning method minimizing the number of shared arcs could be used). The design and flow decisions corresponding to commodities not belonging to a given partition are fixed. Each partition corresponds to a smaller, and hopefully easier to address, problem and is explored by a Local Search heuristic or a more complex metaheuristic, as presented in the previous sections of the chapter. The best solutions obtained within the partitions are collected once the explorations of all partitions are completed, a complete solution is built, a new partition is determined, and the search is restarted. Several approaches may be used to combine the partial solutions into a complete one, e.g., fix to 0 all design arcs

not used in any partition and solve exactly the reduced problem instance (assuming the size has been sufficiently reduced), or fix to 0 as previously and fix to 1 the design arcs with significant flow with respect to capacity (the fixed cost to flow ratio can also be used) and solve the reduced problem. Notice that, similar to all other sequential and parallel metaheuristics, when variable fixing yields a sufficiently small problem instance, an exact MIP solution method may be used to a explore the large neighborhood consisting of all solutions which may be obtained taking the current one as initial solution. We thus define a matheuristic solution method.

Network design requires heavy computation work at each iteration in most cases of interest. Hence, performing many iterations on several space separations may not always appear appropriate and particular care must be taken of the efficiency of the partition exploration. Yet, as the complexity (e.g., time representations and several interrelated combinatorial decision layers, etc.) and dimensions (number of commodities, arcs, time periods, etc.) of the contemplated network design problems continue to grow, decomposition methods appear increasingly needed, in particular combined with other parallelization strategies, cooperation in particular. This defines a rich and challenging research area.

## *7.3 Search-Space Separation: Multi-Search Strategies*

*Independent multi-search* was among the first parallelization strategies proposed. It is also the most simple and straightforward pC parallelization strategy, offering a simple tool for looking for a "good" solution without investment in methodological development or coding. It seeks to accelerate the exploration of the search space by initiating simultaneous solvers from different initial points (with or without different search strategies). No attempt is made to take advantage of the multiple solvers running in parallel other than to identify the best overall solution as a final synchronization step. One therefore needs quite a relatively high number of solvers running in parallel to achieve interesting results.

*Cooperative multi-search* has emerged as one of the most successful metaheuristic methodologies to address hard optimization problems. Cooperative-search strategies go beyond simultaneous runs of multiple independent solvers, and integrate *cooperation mechanisms* to share, *while the search is in progress*, the information obtained from this diversified exploration of the same problem instance. This sharing, and the eventual creation of new information out of the shared one, yields in most cases a collective output of superior quality compared to independent and sequential search, and makes cooperative multi-search a "new" metaheuristic class.

Cooperative-search strategies are defined by the solvers engaged in cooperation and their interaction mechanism, including the nature of the information shared. The metaheuristic or exact solvers do not need to belong to the same solution-method class and may address either the complete problem at hand, or explore partial problems defined by decomposing the initial problem through mathematical

programming or attribute-based heuristic approaches. The decomposition method implicitly defines how a complete solution is built out of partial ones, in the former case. In the latter case, some solvers work on partial problems defined by the particular sets of attributes selected in the decomposition, while others combine the resulting partial solutions into complete solutions to the original problem.

The information-sharing cooperation mechanism specifies how the solvers interact, that is, what information is exchanged and when, how the exchanged information is used globally (if at all), and how each solver acts on the received information, using it within its own search and, thus, transforming it before passing it to other solvers. The goals are to improve the performance of the solvers, and to create a global image, even if not complete and imprecise, of the status of the cooperative search. The status information may then be used to guide solvers, and thus the global search, toward a better performance in terms of solution quality and computational efficiency than the simple concatenation of results obtained by non-cooperating solvers. Exchanged information must be *meaningful* and exchanges must be *timely*.

When and how information is shared specifies the frequency of cooperation activities, who initiates them and when, and whether the concerned solvers must synchronize or not. We do not elaborate further on synchronous communications, partially because of space restrictions, but mostly because synchronization makes parallelism more rigid, and exchanges less timely with respect to the value of new information for solvers. (Notice that most of the discussion on cooperative design applies to synchronous methods as well.) Strategies based on *asynchronous* exchanges thus proved superior to synchronous ones, and are considered as defining the "state-of-the-art" in parallel multi-search metaheuristics. Asynchronous cooperative strategies follow pC/C or pC/KC collegial principles, the main difference being that, "new" knowledge (solutions and more as detailed next) is inferred in the latter case, starting from the information exchanged between solvers.

Asynchronous communications provide the means to build cooperation and information sharing among solvers without incurring synchronization overheads. They also bring adaptability to cooperation strategies, to the extend that the parallel metaheuristic may react more easily and dynamically adapt to the information brought by the other solvers and exploration of the search space, than independent or synchronous search can. These benefits come with potential issues one must care for. For example, the available global-search information available to a given solver may be less "complete" than in a synchronous environment. On the other hand, too frequent data exchanges, combined with simple acceptance rules for incoming information, may induce either a premature solver "convergence" to local optima or an erratic search trajectory similar to a random walk. Hence the interest for applying information-sharing based on quality, meaningfulness, and parsimony principles.

These principles translate into good and diverse solutions being the type of information most often shared, either a single one each time or as a small set. "Good" generally means a local optimum at the end of a search phase, a solution out of an elite set built during the search, or the last solution of an improving sequence of moves. The last case illustrates the parsimony and meaningfulness ideas. One could, clearly, share each improving solution the method identifies. As such solutions are

usually found within a sequence of improving moves, most of them are very similar and, thus, they offer little new information to the other solvers, while significantly disturbing their searches and increasing the communication overload of the parallel metaheuristic. On the other hand, the last improving solution of the sequence brings meaningful information about the status of the search of the emitting solver; it also diversifies the shared information with respect to previous communications. Numerous experiments emphasized the importance of diversification in the shared information. Thus, always selecting and sharing the best available solution out of an elite set proved to rapidly decrease the breath of the search, increase the amount of worthless computational work (many solvers search in the same region), and bring the search to be confined within an often-visited region of the search space, or to an early "convergence" to a not-so-good solution. Strategies that select randomly among an elite set, but bias the choice toward good-and-different solutions, proved more efficient and yielded higher-quality solutions.

*Context* information may also be shared profitably, particularly when embedded in mechanisms used to generate new knowledge or to guide the search. Context refers to data collected by a solver during its exploration, such as the statistical information relative to the presence of particular solution elements in improving solutions (e.g., the medium and long-term memories on selected design arcs or cycles of design arcs built by tabu search), the impact of particular moves on the search trajectory (e.g., the scores of closing/opening design paths within a large adaptive neighborhood search), population diversity measures and individual resilience across generations, etc.

Cooperating solvers may exchange information directly or indirectly. *Direct* exchanges of information, often used in the genetic-based evolutionary literature, occur either when a number of concerned solvers agree on a meeting point in time to share information, or when a solver broadcasts its information to one or several other solvers without prior mutual agreement. The latter case is to be avoided as it requires solvers to include capabilities to store received information without disturbing their own search trajectories until they are ready to consider it. Failure to implement such mechanisms generally results in bad performance, as observed for strategies combining uncontrolled broadcasting of information and immediate acceptance of received data.

*Indirect* exchanges of information are performed through an independent, "centralized", device (implemented on a single processor or on a layered array of processors) called *memory* in this chapter. The memory serves as shared resource of data for cooperating solvers, which access it according to their own internal logic to post and retrieve information. Classical retrieval mechanisms are based on random selection, which may be uniform or, similar to the case of posted information, biased to favor solutions with high rankings based on solution value and diversity. The memory accepts incoming solutions for as long as it is not full. Acceptance becomes conditional to the relative interest of the incoming solution in terms of value, compared to the "worst" solution in the memory, and diversity, a slightly worse solution being preferred if it increases the diversity of solutions in the memory.

To illustrate, consider three pC/C metaheuristics for the MCFND. The first is based on the principles of repetitive applications of Local Search and perturbation of the resulting local optima (e.g., GRASP and Iterated Local Search). Each solver runs a Local Search for the MCFND on the full problem instance for a given initial solution. Solvers deposit their final local optima into the memory, if improved during the current run of the LS, and retrieve a new starting solution (different from the ones already explored) from the memory. The method may be enriched by having each solver run first a Local Search building an elite set of solutions, followed by a Path Relinking on that elite set. Solvers may then send a small group of solutions (e.g., the Local-Search local optima and the best and most diverse solutions yielded by the Path Relinking) to the memory. In a more advanced pC/KC version, the Path Relinking method works also on the solutions in the memory to construct the new starting solution when requested by a solver.

The second pC/C metaheuristic is a memory-based approach for asynchronous parallel Tabu Search, that may be generalized to most neighborhood-based metaheuristics. Each solver runs a TS for the MCFND (see Sect. 4.1.2) from the same or different initial solutions. Solvers could run the same metaheuristic (from different initial solutions, obviously), but it has been shown that running different metaheuristics (different possibly in the value of certain key parameters only) yields a much better overall search. Each solver sends to the memory its local optima, when improved, and imports a solution from the memory before engaging in a diversification phase. The imported solution is selected randomly, biased by rank (in terms of solution value) or by diversity when the selection is made within an elite set. More advanced pC/KC versions are discussed further in the section.

The third illustration is the *Multi-level Cooperative Search*, which proposes a different pC/C asynchronous cooperative strategy based on the controlled diffusion of information. Solvers are arrayed in a linear, conceptually vertical, communication graph and a local memory is associated to each. Each solver works on the original problem but at a different level of aggregation or "coarsening", the first-level solver addressing the complete original problem. It runs a sequential metaheuristic for the MCFND, and communicates exclusively with the two solvers directly above and below, that is, at a higher and a lower aggregation level, respectively. Each solver shares improved solutions, incoming solutions not being transmitted further until modified locally for a number of iterations to enforce the controlled diffusion of information. The local memory is used to receive the information coming from the immediate neighbors, the solver accessing it at moments dynamically determined according to its internal logic (e.g., before diversification). It is noteworthy that one can implement Multi-level Cooperative Search using a centralized memory by adequately defining the communication protocols. Although not yet fully defined and tested, this idea is interesting as it opens the possibility of richer exchange mechanisms combining controlled diffusion and general availability of global information.

Cooperative strategies including mechanisms to create new information and solutions based on the solutions exchanged belong to the p-control knowledge collegial (pC/KC) class. These strategies build on the flexibility of cooperation in

terms of the different metaheuristics and exact methods that can be combined, and on the population of elite solutions being hosted in the centralized memory and continuously enhanced by the cooperating solvers. Mechanisms associated to the memory may thus, e.g., extract and update statistics on the presence of design arcs in the shared solutions (context information on similar statistics within particular solvers may be equally used, when available), possibly with an associated score relative to the solution containing them. Patterns of desirable or undesirable arcs may be thus constructed, and may be evolved to reflect the global status of the search (e.g., initial phase of broad exploration, middle of the solution path starting to reduce the scope of the search, or final phases intensifying in very promising regions before a final choice). Guidance may then be obtained by transmitting arc patterns to solvers, to guide their trajectories toward intensification or diversification phases (enforce the desirability or penalize the selection of certain arcs). Other learning-type mechanisms may be devised (e.g., one may focus on design or commodity paths or trees), this providing a rich field for research. New knowledge may also result from post-optimization applied to solutions in memory, and by generating new solutions out of the elite population in memory by heuristic, genetic, Scatter Search or Path Relinking methods. The new solutions may be returned, on request, to cooperating solvers. They also feed the learning mechanisms associated to the memory.

Historically, two main classes of pC/KC cooperative mechanisms are found in the literature, both based on the idea of exploiting a set of elite solutions, and their attributes, exchanged by cooperating solvers working on the complete problem, but differing in the information kept in memory, *adaptive-memory* and *central-memory*.

*Adaptive-memory* stores and scores partial elements of good solutions and combines them to create new complete solutions that are then improved by the cooperating solvers. The main idea, as initially proposed and applied, is to keep in memory the individual components (e.g., design arcs, or design or commodity paths or trees) making up the elite solutions found by the cooperating solvers, together with memories counting for each component its frequency of inclusion in the best solutions encountered so far, as well as its score, and rank among the population in memory, computed from the attribute values, in particular the objective value of its respective solutions. Solvers construct solutions out of probabilistically selected (biased by rank) solution components in memory, enhance it by Tabu Search or another metaheuristic and deposit their best solutions in the adaptive memory. The probabilistic selection yields, in almost all cases, a new solution made up of components from different elite solutions, thus inducing a diversification effect.

*Central-memory* methods generalize pC/C and adaptive-memory strategies. Complete elite solutions are kept in memory, as well as attributes and context information sent by the solvers involved in cooperation or generated in the central memory. Together, this data is used to create new solutions and knowledge to guide the cooperating solvers and the global search. Solvers may perform constructive, improving and post-optimization heuristics, neighborhood- and population-based metaheuristics and matheuristics, as well as exact solution methods on possibly restricted (through variable fixing) versions of the problem. Other than the informa-

tion received from the cooperating solvers, the central memory keeps newly created information out of the shared data. Statistics-building, information-extraction and learning, and new solution-creation mechanisms provide this new "knowledge". Memories recording the performance of individual solutions, solution components, and solvers may be added to the central memory, and guidance mechanisms based on this knowledge may be gradually built.

We illustrate the basic canvas for central-memory strategies with an early pC/KS metaheuristic combining a genetic solver (Sect. 5.1.2) and several solvers executing the pC/C Tabu Search for the MCFND described above. The TS solvers aggressively explore the search space, building the elite solution set in the memory, while the GA contributes toward increasing the diversity, and hopefully the quality, of the solutions in the memory, which the cooperating TS solvers import. The GA launches once a certain number of elite solutions identified by the TS solvers are recorded in memory, which becomes the initial GA population. Once running, asynchronous migration transfers the best solution of the genetic pool to the memory, as well as solutions from memory toward the genetic population. This strategy was actually implemented and performed well, especially on larger instances. Moreover, it yielded the interesting observation that, while the best overall solution was never found by the GA solver, its inclusion in cooperation allowed the TS solvers to find better solutions, more diversity among solutions in memory translating into a more effective diversification of the global search.

We complete this section by addressing recent developments targeting large or multi-attribute problem settings. The general idea of the new generation of pC/KC meta-heuristics, called *Integrative Cooperative Search* (*ICS*), is to decompose the problem formulation along sets of decision variables to simpler but meaningful problem settings, in the sense that efficient solvers, can be "easily" obtained for the partial problems either by opportunistically using existing high-performing methods or by developing new ones. The decomposition approached evoked relative to domain-decomposition strategies may be used in the ICS context. More complex network structures, e.g., several design layers and time-space network representation, may be decomposed along the layer or time dimensions, respectively. The main components of ICS, to be instantiated for each application, are (1) the decomposition rule; (2) the *Partial Solver Groups* (*PSGs*) addressing the partial problems resulting from the decomposition; (3) the *Integrators* selecting partial solutions from PSGs, combining them, and sending the resulting complete solutions to the *Complete Solver Group* (*CSG*); and (4) the CSG, providing the central memory functionalities of ICS. Notice that, in order to facilitate the cooperation, a unique solution representation, obtained by fixing rather than eliminating variables when defining partial problems, is used throughout ICS. To illustrate consider the generalized network design problem where nodes and arcs are characterized by several attributes (various types, modes, and volumes of capacity, for example) governed by compatibility rules. The problem could then be decomposed along compatible combinations of attributes (a spatial decomposition could be also applied for large networks), Tabu Search solvers could be assigned to each combination (more than one solver could work on each partial problem according to a pC/C

parallel strategy), while a population-based method (or group thereof), Genetic Algorithm, Scatter Search, or Path Relinking, could integrate the partial solutions generated by the TS solvers into complete solutions to the original problem. Monitoring, learning, and guidance activities complement the ICS method. Not much work has been reported in this area for network design, but ICS and related developments make up a promising but challenging research perspective.

## 8 Bibliographical Notes

The use of heuristics to tackle difficult combinatorial optimization problems goes back to the beginnings of operations research. It is therefore not surprising to find some simple heuristics proposed to tackle network design problems, with a clear emphasis on the fixed-charge transportation problem (FCTP). Several papers proposing Local Search methods for the FCTP were published in the late 1960s and the 1970s. Our discussion of Sect. 3.2.2 is based on the paper by Walker (1976), but several other authors proposed methods that exploited similar ideas (Cooper and Drebes 1967; Denzler 1969; Steinberg 1970; Cooper 1975).

From the mid-seventies to the following decade, one saw the rapid development of metaheuristics, which were applied to a large range of combinatorial problems and rapidly became the "go-to approaches" in many circles. While Genetic Algorithms (GAs) came first (Holland 1975), the combinatorial optimization community became keenly aware of the potential of methods that used completely "different" approaches and principles after of the publication of the paper that introduced Simulated Annealing (SA) to a wide audience (Kirkpatrick et al. 1983). This was followed closely by the seminal paper that coined the term *metaheuristics* and presented more formally Tabu Search (TS) (Glover 1986). Other fundamental references on this topic appeared in the following years and laid much of the groundwork for the application of TS (Glover 1989, 1990; Glover and Laguna 1993, 1997) and other metaheuristics, such as Greedy Randomized Adaptive Search Procedure (GRASP) (Feo and Resende 1989, 1995), Variable Neighborhood Search (VNS) (Mladenović and Hansen 1997; Hansen and Mladenović 1999), Path Relinking (PR) and Scatter Search (SS) (Glover 1997; Glover et al. 2000). The case of Iterated Local Search is somewhat stranger since, according to Lourenço et al. (2019), p. 130, "this simple idea has a long history" that goes back to Baxter (1981), but it was formalized systematically in later years. Metaheuristics came of age with the publication in 2003 of the first *Handbook of Metaheuristics* covering a wide range of methods and showing their common points and their successes in the solution of difficult problems (Glover and Kochenberger 2003). Metaheuristics have continued to evolve up to now. An up-to-date introduction to several of the methods covered in this chapter, as well as latest developments, can be found in the latest edition of the Handbook: Tabu Search (Gendreau and Potvin 2019), Simulated Annealing (Delahaye et al. 2019), Variable Neighborhood Search (Hansen et al. 2019), Iterated Local Search (Lourenço et al. 2019), Greedy Randomized Adaptive

Search Procedure (Resende and Ribeiro 2019), Genetic Algorithms (Whitley 2019), and parallel metaheuristics (Crainic 2019). We refer to Silberholz et al. (2019) for a thorough discussion of how to experimentally evaluate the performance of metaheuristics.

Matheuristics were a further development combining advances in algorithm design with the much increased performance of mathematical programming solvers. There is not yet an organized body of literature on the topic, but the seminal paper that proposed the Local Branching method (Fischetti and Lodi 2003) is an excellent introduction to the topic. Furthermore, Local Branching is widely applicable.

After these general comments on solution methods, let us discuss how they were applied to network design problems.

With respect to the FCTP, there are few papers discussing the application of Tabu Search to this problem. The presentation of Sect. 4.1.1 is based on the paper of Sun et al. (1998), which is a standard reference. There are, however, a fairly large number of papers discussing the application of Genetic Algorithms to the problem, but many of these do not deal with the basic version of the problem. Early efforts aimed at using GAs to address the FCTP include the methods proposed by Gottlieb and Paulmann (1998), which examine two possible representations of solutions. Several papers then proposed GAs based on spanning trees to solve the basic FCTP, as well as some of its variants, for instance the FCTP with nonlinear costs (Jo et al. 2007; Hajiaghaei-Keshteli et al. 2010; Molla-Alizadeh-Zavardehi et al. 2014) and the so-called "Step Fixed-Charge Transportation Problem", which is an extension of the FCTP in which different levels of fixed costs may be incurred on an edge depending on the flow that it carries (Molla-Alizadeh-Zavardehi et al. 2014). The presentation of Sect. 5.1.1 refers to the paper by Lofti and Tavakkoli-Moghaddam (2013), which presents a very effective method.

Regarding the various methods for solving the multicommodity capacitated fixed-charge network design problem (MCFND), it became obvious in the early '90's that metaheuristics, such as Tabu Search, were interesting approaches for tackling it. At first, as mentioned in Sect. 2.2, on the basis of the successful applications of TS to complex location problems (see, e.g., Crainic et al. 1993) based on the exploration of the search space of binary location (i.e., design) variables using the add-drop and swap neighborhood structures, it even seemed attractive to contemplate methods based on these neighborhood structures for the MCFND. As we explained in the discussion on neighborhoods, this approach was quite unsatisfactory. The two TS approaches described in Sect. 4.1.2 are based on the papers by Crainic et al. (2000) and Ghamlouche et al. (2003); they were among the best methods available when they were published. Among the other neighborhood-based metaheuristics, we must mention an interesting application of Variable Neighborhood Search to a network design problem with relays (Xiao and Konak 2017).

On the side of population-based methods, the GA described in Sect. 5.1.2 comes from Crainic and Gendreau (1999). It was proposed as part of the cooperative search combining TS and GA solvers described in Sect. 7.

Sections 5.2 and 5.3 present respectively the methods proposed in Ghamlouche et al. (2004) and in Crainic and Gendreau (2007). Another Scatter Search heuristic was proposed in 2005, but for a slightly different version of the MCFND: the undirected variant, in which flows on opposite directions on a given edge must share the same capacity (Alvarez et al. 2005). In this Scatter Search heuristic, solutions are represented as blocks of paths for each origin-destination pair (commodity), such that these paths can handle all the demand for that commodity. The initial pool of solutions is generated using a GRASP procedure. This pool contains, as is usual in Scatter Search implementations, a mixture of good and diverse solutions. The procedure to construct a new solution from a subset taken from the reference set proceeds on a commodity by commodity basis, choosing at each step the block of paths from the solutions in the subset with the lowest cost. A repair procedure is used to restore feasibility. Several strategies are considered for managing the reference set. The method performs well for instances with small numbers of commodities (10 to 50), but performance degrades for larger instances (i.e., 100 commodities).

The improved Scatter Search approach presented in Sect. 5.3.2 has been proposed by Paraskevopoulos et al. (2016). This is the most recent among the various methods that we presented for the MCFND and it yields very good results.

The four matheuristics described in Sect. 6 were first presented, respectively, in Rodríguez-Martin and Salazar-González (2010); Hewitt et al. (2010); Yaghini et al. (2013), and Yaghini et al. (2015). These four methods are competitive and provide the top results. Furthermore, the matheuristic of Hewitt et al. (2010) scales quite well and is capable of handling larger instances than many other approaches. The method that achieves the best results on the benchmark instances is the one by Yaghini et al. (2015). Among sequential methods, it represents the state-of-the-art.

Other recent matheuristics developed for the network design problem with balancing constraints can be found in Vu et al. (2013); Chouman and Crainic (2015). In this problem, an additional constraint that forces both arcs for each pair of arcs $(i, j)$, $(j, i)$ to be either open or closed is added to the model and has a strong impact on the solutions. The first method combines a Tabu Search based on an extension of the cycle-based neighborhood described in 4.1.2, a Path Relinking procedure, which exploits a path-exchange neighborhood, and an exact solution algorithm that is applied to a restriction of the original instance to perform search intensification. The second approach relies on two main elements that can be implemented in the context of a MIP solver: (1) a procedure for generating cutting planes that allows for the efficient computation of tight lower bounds (similar to the approach of Yaghini et al. 2015) , and (2) a variable-fixing procedure to make the resulting restricted problem solvable by a MIP solver in reasonable time. A key feature of the approach is the use of learning mechanisms and memories to record important information about the solution process and to orient it towards promising areas of the solution space. Computational experiments on a large set of test instances confirmed the efficiency of the proposed approach and the high quality of the solutions that it produces.

Before leaving the topic of matheuristics, it is important to mention the large body of litterature that deals with heuristics that rely on mathematical programming

formulations. These include among others the so-called *slope scaling* methods that have been proposed by several authors. These methods are discussed in Chap. 3.

There is a long history of successful developments of parallel heuristics and metaheuristics, a certain number of those contributions targeting network design formulations. A number of surveys, taxonomies, and syntheses present a general view of the topic, including Alba (2005); Alba et al. (2013); Crainic and Hail (2005); Crainic (2008); Crainic and Toulouse (2010); Crainic et al. (2014); Crainic (2019); Cung et al. (2002); Melab et al. (2006); Pedemonte et al. (2011); Schryen (2020); Talbi (2009); Talukdar et al. (2003). Performance measures for parallel metaheuristics are discussed in most of these books, as well as in Barr and Hickman (1993); Crainic and Toulouse (1998, 2003).

All surveys and books include a taxonomy characterizing the parallel strategies for metaheuristics. They agree on most counts, even though terms might differ. Thus, fine- and coarse-grained decomposition, low-level and functional parallelism, domain decomposition, diffusion, as well as synchronous and asynchronous communications are examples of common concepts and terms. While also found broadly, search-space decomposition, multi-search parallelism, and cooperative search may be encountered under different names. Thus, for example, *memory*, *pool*, and *data warehouse* (*reference* and *elite set* are also sometimes used) are equivalent terms found in the parallel metaheuristic literature for the sharing data structures of cooperative search (*blackboard* is often used in the computer-science and artificial-intelligence vocabulary). We use the taxonomy and vocabulary of Crainic and Hail (2005), generalizing Crainic et al. (1996, 1997).

Munguía et al. (2017) proposed a search-space decomposition with restarts matheuristic for the MCFND, which inspired the description of Sect. 7. The authors implement a large neighborhood structure explored with a MIP solver. The Local Search explores each partition defined by a group of commodities, by applying this move (i.e., solving exactly) to each combination of decreasing cardinality of those commodities. Several partitions may be obtained by controlling the graph-partitioning algorithm. A solution-recombination method proceeding in stages (two in the implementation described), by grouping iteratively neighboring partitions (which increases the number of variables one may fix to 0) appears particularly interesting. The authors explored various implementations on different computer architectures and obtained very good results. The commodity-based partitioning and the multi-stage solution-recombination mechanisms are noteworthy as they could be combined with learning mechanisms and cooperative search strategies.

We refer the reader to Crainic (2019) for a detailed discussion relative to cooperative multi-search metaheuristics and the associated literature, and to Crainic et al. (1996, 1997) and Toulouse et al. (1996, 1999a, 2004, 1998, 2000) for the issues related to defining mechanisms and parameters for cooperation, including the quality, meaningfulness, and parsimony principles of information sharing.

Memory-based cooperative pC/C search strategies are described in the literature for most metaheuristic classes and combinatorial optimization problems. To the best of our knowledge, Crainic et al. (1996) were the first to propose a memory-based approach for asynchronous Tabu Search in their study of a multicommodity

location problem with balancing requirements. The proposed method outperformed in terms of solution quality the sequential version as well as several synchronous and broadcast-based asynchronous cooperative strategies. The method was extended to address the MCFND with similar results (Crainic and Gendreau 2002). The illustrations of pC/C multi-search strategies of Sect. 7 are inspired by the work of Ribeiro and Rosseti (2007) on pC/C GRASP for 2-path network design, and the method of Crainic et al. (1996) and Crainic and Gendreau (2002), respectively.

*Multi-level Cooperative Search* (Toulouse et al. 1999b) produced excellent results for various problem settings, including graph and hypergraph partitioning (Ouyang et al. 2000, 2002), feature selection in biomedical data (Oduntan et al. 2008), and covering design (Dai et al. 2009), which are close to network design or may be of interest for parallel metaheuristic strategies (e.g., graph partitioning). The network design illustration of Sect. 7 is based is based on Crainic et al. (2006b).

The many contributions found in the literature show that centralized-memory pC/C asynchronous cooperation strategies are generally offering very good results, yielding high-quality solutions. They are also computationally efficient, with no synchronization overhead, no broadcasting, and no need for complex mechanisms to select the solvers that will receive or send information and to control the cooperation. pC/C strategies have also proved efficient in handling the issue of premature "convergence" in cooperative search, by diversifying the information received by solvers through probabilistic selection from the memory and by a somewhat large and diverse population of solutions in the memory; solvers may thus import different solutions even when their cooperation activities are taking place within a short time span. Such good performances and the availability of shared information kept in the memory has brought the question of whether one could design more advanced cooperation mechanisms taking advantage of the information exchanged among cooperating solvers. The pC/KC strategies are the result of this area of research.

The adaptive-memory terminology was coined by Rochat and Taillard (1995) proposing Tabu Search-based heuristics for vehicle routing. Central-memory methods initiated with the research of Crainic et al. (1996, 1997) for multicommodity location problem with balancing requirements and its extension to the MCFND (Crainic and Gendreau 2002). The basic canvas for central-memory strategies of Sect. 7 comes from those developments, as well as the study of Crainic and Gendreau (1999). The pattern-building learning and guidance mechanisms are inspired by the work of Le Bouthillier et al. (2005) for the vehicle routing problem with time windows (see Le Bouthillier 2007, for a dynamic version of this mechanism).

According to our best knowledge, Crainic et al. (2006a) (see also Di Chiara 2006) were the first to propose an ICS method in the context of designing wireless networks, where seven attributes were considered simultaneously. The proposed pC/KC metaheuristic has Tabu Search solvers address limited subsets of attributes, the others being fixed, and a GA combine the partial solutions generated by solvers into complete solutions to the initial problem. A formal definition and evaluation of ICS (in the vehicle routing context) is to be found in Lahrichi et al. (2015).

## 9 Conclusions and Perspectives

One of the main things that comes to mind when considering heuristic solution procedures for network design problems is the realization that there has been a very steady improvement in the performance of these methods over time: state-of-the-art methods are now capable of producing solutions that are quite close to optimal ones in a fraction of the time required by exact approaches. This can be very important in some practical settings. It is also interesting to note that some of the most recent methods scale up rather well, which means that they are able to tackle instances of significant size, similar to the ones that one is likely to encounter in many practical applications.

These conclusions can be traced, in our opinion, to three main causes. The first is the proper exploitation of the mathematical properties of optimal solutions of fixed-charge problems, namely the fact that, in most cases, they can be found at extreme points of the feasible set. This property has had a deep impact on the definition of search spaces for many methods, making these much more effective. The second reason is the amazing developments in the performance of MIP solvers, which has allowed for the integration of dedicated exact solution procedures within many metaheuristics and matheuristics. The third reason, but not the least, are the advances in the design of solution procedures, which now display a sophisticated architecture to deal with the various challenges of network design problems. It is the combination of these three factors that explain where we stand now in our efforts to tackle these problems.

This having been said, it is important to state that a lot remains to be done: there are indeed several intriguing approaches that one might wish to consider for tackling network design problems. Among these, the Unified Hybrid Genetic Search (UHGS), which combines the exploitation of a carefully managed population of solutions with neighborhood-based search, immediately comes to mind considering the successes obtained by this method on a wide range of difficult vehicle routing problems (see, e.g., Vidal et al. 2012, 2014). Combining the ideas of UHGS with tailored exact solution procedures could lead to very powerful matheuristics. A second fascinating perspective is that of parallel and cooperative search, particularly in the ICS version. Indeed, combining various decomposition strategies, along different dimensions of design problems, with state-of-the-art matheuristics for the resulting problems and the recombination of solutions presents fascinating challenging and very promising perspectives. This is also an area where advanced learning mechanisms find a natural niche to extend the classical memory-based capabilities of metaheuristics.

Last but not least, there is the challenge of metaheuristics and matheuristics for the many variants of network design problems brought by applications. The other chapters of this book present these models and identify the associated algorithmic challenges and perspectives.

## References


Alba, E. (Ed.) (2005). *Parallel metaheuristics: A new class of algorithms*. Hoboken, NJ: Wiley.

Alba, E., Luque, G., & Nesmachnow, S. (2013). Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research, 20*(1), 1–48.

Alvarez, A. M., González-Velarde, J. L., & De-Alba, K. (2005). Scatter search for network design problem. *Annals of Operations Research, 138*, 159–178.

Barr, R. S., & Hickman, B. L. (1993). Reporting computational experiments with parallel algorithms: issues, measures, and experts opinions. *ORSA Journal on Computing, 5*(1), 2–18.

Baxter, J. (1981). Local optima avoidance in depot location. *Journal of the Operational Research Society, 32*, 815–819.

Chouman, M., & Crainic, T. G. (2015). Cutting-plane matheuristic for service network design with design-balanced requirements. *Transportation Science 49*(1), 99–113.

Chouman, M., Crainic, T. G., & Gendron, B. (2009). A cutting-plane algorithm for multi-commodity capacitated fixed charge network design. Tech. Rep. CIRRELT-2009-20, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et les transports, Université de Montréal, Montréal, QC, Canada.

Cooper, L. (1975). The fixed charge problem-I: A new heuristic method. *Computers & Mathematics with Applications, 1*, 89–95.

Cooper, L., & Drebes, C. (1967). An approximate solution method for the fixed charge problem. *Naval Research Logistics Quarterly, 14*, 101–113.

Crainic, T. G. (2008). Parallel solution methods for vehicle routing problems. In B. L.Golden, S. Raghavan, & E. A. Wasil (Eds.). *The vehicle routing problem: latest advances and new challenges* (pp. 171–198). New York, NY: Springer.

Crainic, T. G. (2019). Parallel metaheuristics and cooperative search. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (3rd ed., pp. 419–451). Berlin: Springer.

Crainic, T.G., & Gendreau, M. (1999). Towards an evolutionary method—cooperating multi-thread parallel tabu search hybrid. In S. Voß, S. Martello, C. Roucairol, & I. H. Osman (Eds.), *Meta-heuristics 98: Theory and applications* (pp. 331–344). Norwell, MA: Kluwer Academic Publishers.

Crainic, T.G., & Gendreau, M. (2002). Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics, 8*(6), 601–627.

Crainic, T.G., & Gendreau, M. (2007) A scatter search heuristic for the fixed-charge multicommodity flow network design problem. In K. Doerner, M. Gendreau, P. Greistorfer, W. J. Gutjahr, R. F. Hartl, & M. Reimann (Eds.), *Metaheuristics—progress in complex systems optimization* (pp. 25–40). New York, NY: Springer.

Crainic, T. G., & Hail, N. (2005). Parallel meta-heuristics applications. In E. Alba (Ed.), *Parallel metaheuristics: A new class of algorithms* (pp. 447–494). Hoboken, NJ: Wiley

Crainic, T. G., & Toulouse, M. (1998). Parallel metaheuristics. In T. G. Crainic, & G. Laporte (Eds.), *Fleet management and logistics* (pp. 205–251). Norwell, MA: Kluwer Academic Publishers.

Crainic, T. G., & Toulouse, M. (2003). Parallel strategies for meta-heuristics. In F. Glover, & G. Kochenberger (Eds.), *Handbook in metaheuristics* (pp. 475–513). Norwell, MA: Kluwer Academic Publishers.

Crainic, T. G., & Toulouse, M. (2010) Parallel meta-heuristics. In M. Gendreau, J.-Y. Potvin (Eds.). *Handbook of metaheuristics* (2nd ed., pp. 497–541). Berlin: Springer.

Crainic, T. G., Gendreau, M., Soriano, P., & Toulouse, M. (1993). A Tabu search procedure for multicommodity location/allocation with balancing requirements. *Annals of Operations Research, 41*, 359–383.

Crainic, T. G., Toulouse, M., & Gendreau, M. (1996). Parallel asynchronous Tabu search for multicommodity location-allocation with balancing requirements. *Annals of Operations Research, 63*, 277–299.

Crainic, T. G., Toulouse, M., & Gendreau, M. (1997). Towards a taxonomy of parallel tabu search algorithms. *INFORMS Journal on Computing, 9*(1), 61–72.

Crainic, T. G., Gendreau, M., & Farvolden, J. M. (2000). A simplex-based Tabu search method for capacitated network design. *INFORMS Journal on Computing, 12*(3), 223–236.

Crainic, T. G., Di Chiara, B., Nonato, M., & Tarricone, L. (2006a) Tackling electrosmog in completely configured 3G networks by parallel cooperative meta-heuristics. *IEEE Wireless Communications, 13*(6), 34–41.

Crainic, T. G., Li, Y., & Toulouse, M. (2006b). A first multilevel cooperative algorithm for the capacitated multicommodity network design. *Computers & Operations Research, 33*(9), 2602–2622.

Crainic, T. G., Davidović, T., & Ramljak, D. (2014). Designing parallel meta-heuristic methods. In M. Despotovic-Zrakic, V. Milutinovic, & A. Belic (Eds.), *High performance and cloud computing in scientific research and education* (pp. 260–280). Hershey, PA: IGI Global

Cung, V. D., Martins, S. L., Ribeiro, C. C., & Roucairol, C. (2002). Strategies for the parallel implementations of metaheuristics. In C. Ribeiro, P. Hansen (Eds.), *Essays and surveys in metaheuristics* (pp. 263–308). Norwell, MA: Kluwer Academic Publishers.

Dai, C., Li, B., & Toulouse, M. (2009). A multilevel cooperative Tabu search algorithm for the covering design problem. *Journal of Combinatorial Mathematics and Combinatorial, Computing 68*, 35–65.

Delahaye, D., Chaimatanan, S., & Mongeau, M. (2019) Simulated annealing: From basics to applications. In M. Gendreau, J.-Y. Potvin (Eds.), *Handbook of metaheuristics* (3rd ed., pp 1–35). Cham: Springer.

Denzler, D. R. (1969). An approximate algorithm for the fixed charge problem. *Naval Research Logistics Quarterly, 16*, 411–416.

Di Chiara, B. (2006). Optimum planning of 3G cellular systems: Radio propagation models and cooperative parallel meta-heuristics. PhD thesis, Dipartimento di ingegneria dell'innovatione, Universitá degli Studi di Lecce, Lecce, Italy.

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters, 8*, 67–71.

Feo, T. A., & Resende, M. G. C. (1995) Greedy randomized adaptive search procedures. *Journal of Global Optimization, 6*, 109–134.

Fischetti, M., & Lodi, A. (2003). Local branching. *Mathematical Programming Series B, 98*, 23–47.

Gendreau, M., & Potvin, J.-Y. (2019). Tabu search. In M. Gendreau, J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (3rd ed., pp. 37–55). Cham: Springer.

Gendreau, M., Hertz, A., & Laporte, G. (1994). A Tabu search heuristic for the vehicle routing problem. *Management Science, 40*(10), 1276–1290.

Gendron, B., & Crainic, T. G. (1994). Relaxations for multicommodity network design problems. Publication CRT-965, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada.

Ghamlouche, I., Crainic, T. G., & Gendreau, M. (2003). Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Operations Research, 51*(4), 655–667.

Ghamlouche, I., Crainic, T. G., & Gendreau, M. (2004). Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations Research, 131*, 109–133.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research, 1*(3), 533–549

Glover, F. (1989). Tabu search—part I. *ORSA Journal on Computing, 1*(3), 190–206.

Glover, F. (1990). Tabu search—part II. *ORSA Journal on Computing, 2*(1), 4–32.

Glover, F. (1997). A template for scatter search and path relinking. In J. Hao, E. Lutton, E. Ronald, M. Schoenauer, & D. Snyers (Eds.), *Artificial evolution, lecture notes in computer science* (Vol. 1363, pp. 13–54). Berlin: Springer.

Glover, F., & Kochenberger, G. (Eds.) (2003). *Handbook of metaheuristics*. Norwell, MA: Kluwer Academic Publishers.

Glover, F., & Laguna, M. (1993). Tabu search. In C. Reeves (Ed.), *Modern heuristic techniques for combinatorial problems* (pp. 70–150). Oxford: Blackwell Scientific Publications.

Glover, F., & Laguna, M. (1997) *Tabu search*. Norwell, MA: Kluwer Academic Publishers.

Glover, F., Laguna, M., & Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics, 39*(3), 653–684.

Gottlieb, J., & Paulmann, L. (1998). Genetic algorithms for the fixed charge transportation problem. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation* (pp. 330–335)

Hajiaghaei-Keshteli, M., Molla-Alizadeh-Zavardehi, S., & Tavakkoli-Moghaddam, R. (2010) Addressing a nonlinear fixed charge transportation problem using a spanning tree based genetic algorithm. *Computers & Industrial Engineering, 59*, 259–271.

Hansen, P., Mladenović, N. (1999). An introduction to variable neighborhood search. In S. Voß, S. Martello, C. Roucairol, & I. H. Osman (Eds.), *Meta-heuristics 98: Theory & applications* (pp. 433–458). Norwell, MA: Kluwer Academic Publishers.

Hansen, P., Mladenović, N., Brimberg, J., Pérez, J. A. M. (2019). Variable neighborhood search. In: M. Gendreau, J.-Y. Potvin (Eds.), *Handbook of metaheuristics* (3rd ed., pp. 57–97). Cham: Springer.

Hewitt, M., Nemhauser, G. L., & Savelsbergh, M. W. (2010) Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing, 22*, 314–325.

Hirsch, W. M., & Dantzig, G. B. (1954) *Notes on linear programming part XIX, the fixed charge problem. Memorandum* (Vol. 1383). Santa Monica, CA: Rand Research

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press

Jo, J. B., Li, Y., & Gen, M. (2007). Nonlinear fixed charge transportation problem by spanning tree-based genetic algorithm. *Computers & Industrial Engineering, 53*, 290–298.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983) Optimization by simulated annealing. *Science, 220*, 671–680.

Lahrichi, N., Crainic, T. G., Gendreau, M., Rei, W., Crisan, G. C., & Vidal, T. (2015) An integrative cooperative search framework for multi-decision-attribute combinatorial optimization. *European Journal of Operational Research, 246*(2), 400–412.

Le Bouthillier, A. (2007). Recherches coopératives pour la résolution de problèmes d'optimisation combinatoire. PhD thesis, Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, QC, Canada.

Le Bouthillier, A., Crainic, T. G., & Kropf, P. (2005). A guided cooperative search for the vehicle routing problem with time windows. *IEEE Intelligent Systems, 20*(4), 36–42.

Lofti, M. M., & Tavakkoli-Moghaddam, R. (2013). A genetic algorithm using priority-based encoding with new operators for fixed charge transportation problems. *Applied Soft Computing, 13*(5), 2711–2726.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2019). Iterated local search: Framework and applications. In: M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of metaheuristics* (3rd ed., pp. 129–168). Cham: Springer.

Melab, N., Talbi, E. G., Cahon, S., Alba, E., & Luque, G. (2006). Parallel metaheuristics: Models and frameworks. In E.L. Ghazali Talbi (Ed.), *Parallel combinatorial optimization* (pp. 149–162). New York, NY: Wiley.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research, 24*:1097–1100

Molla-Alizadeh-Zavardehi, S., Mahmoodirad, A., & Rahimian, M. (2014). Step fixed charge transportation problems. *Indian Journal of Science and Technology, 7*(7), 949–954.

Munguía, L. M., Ahmed, S., Bader, D. A., Nemhauser, G. L., Goel, V., & Shao, Y. (2017). A parallel local search framework for the fixed-charge multicommodity network flow problem. *Computers & Operations Research, 77*, 44–57.

Oduntan, I. O., Toulouse, M., Baumgartner, R., Bowman, C., Somorjai, R., & Crainic, T. G. (2008). A multilevel Tabu search algorithm for the feature selection problem in biomedical data sets. *Computers & Mathematics with Applications, 55*(5), 1019–1033.

Ouyang, M., Toulouse, M., Thulasiraman, K., Glover, F., & Deogun, J. S. (2000). Multi-level cooperative search: Application to the netlist/hypergraph partitioning problem. In *Proceedings of International Symposium on Physical Design* (pp. 192–198). New York, NY: ACM Press.

Ouyang, M., Toulouse, M., Thulasiraman, K., Glover, F., & Deogun, J. S. (2002). Multilevel cooperative search for the circuit/hypergraph partitioning problem. *IEEE Transactions on Computer-Aided Design, 21*(6), 685–693.

Paraskevopoulos, D. C., Bektaş, T., Crainic, T. G., & Potts, C. N. (2016). A cycle-based evolutionary algorithm for the fixed-charge capacitated multi-commodity network design problem. *European Journal of Operational Research, 253*(1), 265–279.

Pedemonte, M., Nesmachnow, S., & Cancela, H. (2011). A survey of parallel ant colony optimization. *Applied Soft Computing, 11*(8), 5181–5197.

Resende, M. G. C., & Ribeiro, C. C. (2019). Greedy randomized adaptive search procedures: Advances and extensions. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (3rd ed., pp. 169–220). Cham: Springer.

Ribeiro, C. C., & Rosseti, I. (2007). Efficient parallel cooperative implementations of GRASP heuristics. *Parallel Computing, 33*(1), 21–35.

Rochat, Y., & Taillard, E. D. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics, 1*(1), 147–167.

Rodríguez-Martin, I., & Salazar-González, J. J. (2010). A local branching heuristic for the capacitated fixed-charge network design problem. *Computers & Operations Research, 37*, 575–581.

Schryen, G. (2020). Parallel computational optimization in operations research: A new integrative framework, literature review and research directions. *European Journal of Operational Research, 287*(1), 1–18.

Silberholz, J., Golden, B., Gupta, S., & Wang, X. (2019). Computational comparison of metaheuristics. In M. Gendreau, & J.-Y. Potvin (Eds.). *Handbook of metaheuristics* (3rd ed., pp. 581–604). Cham: Springer.

Steinberg, D. I. (1970). The fixed charge problem. *Naval Research Logistics Quarterly, 17*, 217–236.

Sun, M., Aronson, J. E., McKeown, P. G., & Drinka, D. (1998). A Tabu search procedure for the fixed charge transportation problem. *European Journal of Operational Research, 106*, 441–446.

Talbi, E. G. (Ed.) (2009). *Metaheuristics: From design to implementation*. Hoboken, NJ: Wiley.

Talukdar, S., Murthy, S., & Akkiraju, R. (2003). Assynchronous teams. In F. Glover, & G. Kochenberger (Eds.), *Handbook in metaheuristics* (pp. 537–556) Norwell, MA: Kluwer Academic Publishers.

Toulouse, M., Crainic, T. G., & Gendreau, M. (1996). Communication issues in designing cooperative multi thread parallel searches. In I. H. Osman, & J. P. Kelly (Eds.), *Meta-heuristics: Theory & applications* (pp. 501–522). Norwell, MA: Kluwer Academic Publishers.

Toulouse, M., Crainic, T. G., Sansó, B., & Thulasiraman, K. (1998). Self-organization in cooperative search algorithms. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics* (pp. 2379–2385). Madisson, WI: Omnipress.

Toulouse, M., Crainic, T. G., & Sansó, B. (1999a). An experimental study of systemic behavior of cooperative search algorithms. In S. Voß, S. Martello, C. Roucairol, & I. H. Osman (Eds.), *Meta-heuristics 98: Theory & applications* (pp. 373–392). Norwell, MA: Kluwer Academic Publishers.

Toulouse, M., Thulasiraman, K., & Glover, F. (1999b). Multi-level cooperative search: a new paradigm for combinatorial optimization and an application to graph partitioning. In P. Amestoy, P. Berger, M. Daydé, I. Duff, V. Frayssé, L. Giraud, & D. Ruiz (Eds.), *Fifth International Euro-Par Parallel Processing Conference, Lecture Notes in Computer Science* (Vol. 1685, pp. 533–542). Heidelberg: Springer.

Toulouse, M., Crainic, T. G., & Thulasiraman, K. (2000). Global optimization properties of parallel cooperative search algorithms: A simulation study. *Parallel Computing, 26*(1), 91–112.

Toulouse, M., Crainic, T. G., & Sansó, B. (2004). Systemic behavior of cooperative search algorithms. *Parallel Computing, 30*(1), 57–79.

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems. *Operations Research, 60*(3), 611–624.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research, 234*(3), 658–673.

Vu, D. M., Crainic, T. G., & Toulouse, M. (2013). A three-stage matheuristic for the capacitated multi-commodity fixed-cost network design with design-balance constraints. *Journal of Heuristics, 19*, 757–795.

Walker, W. E. (1976). A heuristic adjacent extreme point algorithm for the fixed charge problem. *Management Science, 22*, 587–596.

Whitley, D. (2019). Next generation genetic algorithms: A user's guide and tutorial. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of metaheuristics* (3rd ed., pp. 245–274). Cham: Springer.

Xiao, Y., & Konak, A. (2017). A variable neighborhood search for the network design problem with relays. *Journal of Heuristics, 23*, 137–164.

Yaghini, M., Rahbar, M., & Karimi, M. (2013). A hybrid simulated annealing and column generation approach for capacitated multicommodity network design. *Journal of the Operational Research Society, 64*, 1010–1020.

Yaghini, M., Karimi, M., Rahbar, M., Sharifitabar, H. (2015). A cutting-plane neighborhood structure for fixed-charge capacitated multicommodity network design problem. *INFORMS Journal on Computing, 27*(1), 48–58.