International Series in
Operations Research & Management Science

Ghaith Rabadi *Editor*

Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling

Springer

International Series in Operations Research & Management Science

Volume 236

More information about this series at http://www.springer.com/series/6161

Ghaith Rabadi
Editor

Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling

Springer

Editor
Ghaith Rabadi
Engineering Management
and Systems Engineering
Old Dominion University
Norfolk, VA, USA

ISSN 0884-8289 ISSN 2214-7934 (electronic)
International Series in Operations Research & Management Science
ISBN 978-3-319-26022-8 ISBN 978-3-319-26024-2 (eBook)
DOI 10.1007/978-3-319-26024-2

Library of Congress Control Number: 2015954630

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

To the most important women in my life:
Shereen, Celine, Renee, Karoline, and Lana.

Introduction

Planning and scheduling research has started at the early years of Operation Research (OR) more than 50 years ago. Since then, people have been motivated and challenged by the complexity of planning and scheduling problems. Most of the research in this field is commonly published in various theoretical and applied OR journals, which are also home to many other OR topics. Traditionally, planning and scheduling problems are a class of discrete optimization problems, which have mostly been addressed by linear and nonlinear mathematical programming models. Despite all the progress achieved in mathematical programming methods and the advancement in the computer technology, the vast majority of practical planning and scheduling problems are NP-complete/NP-hard which are time-consuming to solve via mathematical programming due to their computationally explosive nature. Hence, the development in inexact methods and heuristics has been instrumental to solving such problems especially for large instances.

The scope of this book is limited to heuristics, metaheuristics, and approximate methods as applied to planning and scheduling problems. While it is not possible to give a comprehensive treatment of this topic in one book, the aim of this work is to provide the reader with a diverse set of planning and scheduling problems and different heuristic approaches to solve them. The problems range from traditional single stage and parallel machine problems to more modern settings such as robotic cells and flexible job shop networks. Furthermore, some chapters deal with deterministic problems, while some others treat stochastic problems. Unlike most of the literature that deals with planning and scheduling problems in the manufacturing and production environments, in this book the environments were extended to nontraditional applications such as spatial scheduling (optimizing space over time), runway scheduling, and surgical scheduling. The solution methods used in the different chapters of the book also spread from well-established heuristics and metaheuristics such as Genetic Algorithms, Simulated Annealing, and Tabu Search to more recent ones such as Ant Colony Optimization, Meta-RaPS, and Worm Optimization.

None of the chapters has been previously published in its totality and each chapter offers a contribution to the body of the knowledge by presenting results and insights from applying heuristics, metaheuristics, or approximate methods to complex, recent, and diverse problems.

Norfolk, VA, USA Ghaith Rabadi

Contents

Contributors

Ghaith Rabadi Department of Engineering Management and Systems Engineering, Old Dominion University, Norfolk, VA, USA

Mohammed Al-Salem Department of Mechanical and Industrial Engineering, College of Engineering, Qatar University, Doha, Qatar

Arif Arin Department of Business Administration, Middle East Technical University, Universiteler Mahallesi, Ankara, Turkey

Jean-Paul Arnaout Gulf University for Science and Technology, Kuwait City, Kuwait

Nixon Dcoutho Department of Industrial and Systems Engineering, Northern Illinois University, DeKalb, IL, USA

Farbod Farhadi Roger Williams University, Bristol, RI, USA

J. Bennett Foster DuPont Company, Wilmington, DE, USA

Christopher Garcia College of Business, University of Mary Washington, Fredericksburg, VA, USA

Gulsah Hancerliogullari Department of Industrial Engineering, Faculty of Engineering and Natural Sciences, Istanbul Bilgi University, Istanbul, Turkey

Kadir Oymen Hancerliogullari Department of Pediatric Surgery, Faculty of Medicine, Giresun University, Giresun, Turkey

Mohamed Haouari Department of Mechanical and Industrial Engineering, College of Engineering, Qatar University, Doha, Qatar

Wael Khallouli Department of Mechanical and Industrial Engineering, College of Engineering, Qatar University, Doha, Qatar

Mohamed Kharbeche Qatar Road Safety Studies Center, Qatar University, Doha, Qatar

Peter L. King Lean Dynamics, Inc., Newark, DE, USA

Emrah Koksalmis Department of Industrial Engineering, Faculty of Management, Istanbul Technical University, Istanbul, Turkey

Stefan Lausch University of Hagen, Hagen, Germany

Lingrui Liao Grado Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, VA, USA

Po-Chen Lin Edward P. Fitts Department of Industrial and Systems Engineering, 300 Daniels Hall North Carolina State University, Raleigh, NC, USA

Lars Mönch University of Hagen, Hagen, Germany

Reinaldo J. Moraga Department of Industrial and Systems Engineering, Northern Illinois University, DeKalb, IL, USA

Subhash C. Sarin Grado Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, VA, USA

Hanif D. Sherali Grado Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, VA, USA

Bulent Soykan Department of Engineering Management and Systems Engineering, Old Dominion University, Norfolk, VA, USA

Reha Uzsoy Edward P. Fitts Department of Industrial and Systems Engineering, 300 Daniels Hall North Carolina State University, Raleigh, NC, USA

Amrusha Varadarajan Grado Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, VA, USA

Chapter 1
Approximation Algorithms for Spatial Scheduling

Christopher Garcia and Ghaith Rabadi

Abstract Spatial scheduling problems involve a set of jobs which have spatial dimensions in addition to traditional scheduling considerations such as due dates or processing times. In these problems processing space is a limited resource, and the scheduling decisions must determine both when and where the each job will be processed as well as each job's layout orientation. Spatial scheduling problems find many real-world applications in industries such as shipbuilding and aircraft assembly, where there is limited working space available and tasks utilize significant amounts of spatial resources in their completion. In this chapter we discuss spatial scheduling and present several heuristic and metaheuristic algorithms for this problem class.

Keywords Spatial scheduling • Greedy algorithms • Metaheuristics • MetaRaPS

1.1 Introduction to Spatial Scheduling

Scheduling problems typically involve a set of jobs where each job has a due date, a set of resource requirements (such as processing time or manpower requirements), and a set of costs or profits. In spatial scheduling, physical space is viewed as a limited resource and each job requires a certain amount of two-dimensional space. Moreover, space is regarded as continuous (or approximately continuous) which generally prohibits modeling spatial scheduling problems as simple assignment problems. Spatial scheduling problems generally arise when there is a limited

C. Garcia (✉)
College of Business, University of Mary Washington, Fredericksburg, VA 22407, USA
e-mail: cgarcia@umw.edu

G. Rabadi
Department of Engineering Management and Systems Engineering,
Old Dominion University, Norfolk, VA 23529, USA
e-mail: grabadi@odu.edu

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_1

amount of working space available and when each job requires a certain amount of space for completion. Spatial scheduling problems are frequently encountered in shipbuilding, aircraft assembly, and heavy industries.

There is a relatively small amount of literature on spatial scheduling compared to other types of scheduling. Much of the applied spatial scheduling literature addresses two related problems in shipbuilding: the *block assembly problem* and the *block painting problem*. These problems both involve assigning polygonal objects to specific work areas while simultaneously determining the start times and locations within the assigned work area for each job. The block assembly problem was addressed in Cho et al. (2001). In addition to spatial aspects, this problem also has a workload balancing objective. They developed an approach involving an operation strategy algorithm along with separate algorithms for block scheduling, arrangement, and assignment. More recently, Caprace et al. (2013) applied a 3D guided-search bin-packing algorithm to solve a similar spatial scheduling problem within a shipyard. The block assembly problem was addressed in Lee and Lee (1996) through the use of an expert system based on backtracking search and adjustment rules. A related problem involving pre-scheduled jobs for which spatial locations had to be determined was addressed by Varghese and Yoon (2005). They developed a genetic algorithm for solving this problem.

In addition to specific applications, work has been done on more generalized spatial scheduling models and related problems. Padberg (2000) examined the problem of packing smaller boxes into a larger one. Because spatial scheduling problems can be viewed as having two spatial dimensions and one time dimension, this work provides an important conceptual foundation for spatial scheduling. Important theoretical work in Duin and Van Sluis (2006) and Paulus and Hurink (2006) showed that in general, scheduling problems which involve spatial constraints are NP-hard. A generalized spatial scheduling model involving rectangular jobs with multiple non-identical rectangular work areas was first addressed in Garcia (2010). This problem class involved jobs with release times and due dates, and the objective studied was to minimize the total tardiness. Integer programming (IP) models were given for single and multiple-area versions of the problem and metaheuristics based on simulated annealing and local search were developed. In Garcia and Rabadi (2011), an algorithm based on the Metaheuristic for Randomized Priority Search (Meta-RaPS) was developed for this problem class, and in Garcia and Rabadi (2013) a fast non-iterative heuristic was developed. More recently, Srinivasan (2014) studied a similar problem involving rectangular jobs and a single rectangular processing area with the objective of minimizing the sum of completion times. IP formulations were developed along with two heuristic algorithms. Additionally, this work provided insights into when batching is advantageous and how it should be applied.

In this chapter we show how heuristic and metaheuristic approximation algorithms can be effectively applied to spatial scheduling problems. We use the multiple parallel work area problem studied in Garcia and Rabadi (2013) as our representative problem class. This involves rectangular jobs which have release times and due dates and can be rotated 90°, multiple non-identical processing

areas, and an objective of minimizing the total tardiness. This problem class was selected because it generalizes many other spatial scheduling problem classes. For example, single-workspace problems simply use one processing area, fixed orientations simply prohibit rotations, and the absence of due dates or release times can be modeled by setting these values to zero for all jobs. Additionally, other objective functions may also easily be incorporated. We present several heuristic and metaheuristic approximation algorithms from the literature for this problem class and provide a comparative analysis on a set of benchmark problems.

1.2 General Problem Formulation

The general spatial scheduling problem may be stated as follows. There is a set of n jobs each having a width w_j, height h_j, processing time (duration) p_j, due date d_j, and release time (or earliest start date) e_j. Each job must be processed inside a rectangular processing area k, and there are m available processing areas each having a width W_k and height H_k. Jobs may be processed in parallel and any job may be assigned to any processing area, provided it can fit inside. Furthermore, jobs may be rotated by 90° as needed. The objective is to assign each job a start time s_j, a processing area, and a location inside the assigned processing area together with an orientation (i.e., one of two possible 90° rotations) in order to minimize the total tardy time. The tardiness for job j is defined as $T_j = \max\left(0, C_j - d_j\right)$ where C_j is the job's completion time and d_j is its due date. The objective will then be $\sum_{j=1}^{n} T_j$ where n is the total number of jobs. A depiction of the scheduling of jobs within an area is shown in Fig. 1.1 (taken from Garcia and Rabadi 2013).

It is seen in Fig. 1.1 that each job is assigned a specific location and orientation inside a processing area, and remains there for the duration of its processing time. A general IP formulation for this problem may be found in Garcia (2010).

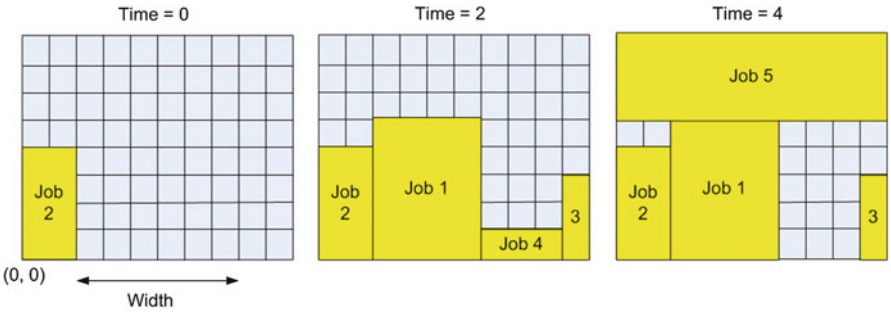


Fig. 1.1 Scheduling jobs within a processing area

1.3 Feasible Schedule Construction: The Bottom-Left Time-Incrementing Heuristic

Under the general problem formulation there are multiple non-identical processing areas which may be used in parallel. This use of multiple processing areas adds a significant amount of complexity to the problem, as it requires assigning a processing area to each job in addition to assigning a location and orientation within the assigned area. In the sections that follow, several algorithms are developed which use separate steps to first assign processing areas to the jobs and then assign start times, locations, and orientations to each job. In this section we restrict our attention to the sub-problem of constructing a feasible schedule for a single area given a list of jobs assigned to that area. The resulting algorithm will then be used as a key ingredient for the algorithms presented in subsequent sections.

A constructive method called the Bottom-Left-Time-Incrementing heuristic (BLTI) was first developed in Garcia (2010) in order to generate a feasible schedule for a single processing area. BLTI combines the well-known Bottom-Left (BL) strip-packing heuristic developed by Baker et al. (1980) with a time-incrementing mechanism for schedule construction. BL is a greedy rectangle-packing method that places a set of rectangles into a strip or larger rectangle. BL sequentially packs rectangles into the strip in the most bottom-left location where they fit. BLTI employs BL but modifies it with a current time variable that is incremented as jobs enter and leave the processing area. BLTI schedules jobs to the area in the order in which they are given. Because the current time variable is always incremented and never decremented, BLTI is guaranteed to construct feasible schedules regardless of the input order. However, the schedule quality (in terms of the objective function) is entirely dependent upon the input ordering of the jobs. The BLTI algorithm is shown in Algorithm 1 below. It is assumed that each job in the input list can fit inside the processing area.

Algorithm 1: BLTI with inputs *Area* and *Jobs*

Procedure BLTI(*Area*, *Jobs* $= 1 \ldots n$) DO:
  CurrentTime$\leftarrow 0$
  For each j in *Jobs*:
    CurrentTime$\leftarrow$ MAX(*CurrentTime*, e_j)
      Remove each job k from *Area* where $s_k + p_k \leq$ *CurrentTime*
      $(X,Y) \leftarrow$ Pack j into *Area* using BL
      While $(X,Y) =$ NIL:
        Select job k from *Area* with smallest $s_k + p_k$
        CurrentTime$\leftarrow s_k + p_k$
        Remove job k from *Area*
        $(X,Y) \leftarrow$ Pack j into *Area* using BL
      $(x_j, y_j) \leftarrow (X,Y)$
      $s_j \leftarrow$ *CurrentTime*

In Algorithm 1, (x_j, y_j) designates the x- and y-coordinates assigned to the bottom-left corner of job j. The BL packing procedure returns the bottom-left coordinate of the next rectangle added, or NIL if this is not possible. When BLTI attempts to pack a job into the processing area using BL, it is assumed to try both orientations and pack the job using the first orientation which fits. In cases where rotations are not permitted, this can easily be modified to restrict packing only for the specified orientation.

1.4 The Packing Factor Heuristic

A fast, non-iterative heuristic algorithm for the multiple-area spatial scheduling problem described above was developed in Garcia and Rabadi (2013), which we present in this section. This algorithm, referred to as the *Packing Factor Heuristic* (PFH), consists of three basic components: (1) an area assignment heuristic, (2) BLTI for constructing feasible schedules for a single area at a time, and (3) a set of input ordering rules. The complete scheduling algorithm executes a sequence of four main steps:

1. A heuristic assigns each job to an area where it should be processed.
2. For each area, several different job input sequences are generated for the area's assigned jobs.
3. BLTI is used to construct candidate schedules for each area separately based on the input sequences generated in step 2. The schedule resulting in the least amount of total tardiness is retained for each area.
4. Since jobs are processed independently in the different areas, the resulting schedules for each area are simply concatenated together to form the master schedule.

1.4.1 Area Assignment Heuristic

The first and most complex step of this scheduling algorithm is to assign jobs to processing areas. The general area assignment strategy aims to place jobs that will likely have the least flexibility of location first while keeping the aggregated space consumption fraction as small as possible within each area. In a purely two-dimensional packing problem it is easy to see that larger rectangles generally have less flexibility as to where they can be placed. Packing rectangles in a largest-first order is an intuitive strategy which was also shown to be effective in the earlier strip-packing literature (Hopper and Torton 2001). However, it is not as easy to see what affords one job more flexibility than another in the type of spatial scheduling problem under consideration. While a job may take up a significant amount of 2D space, it may also have a large amount of slack time in which it may be

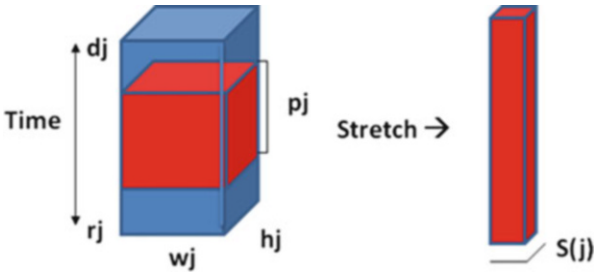


Fig. 1.2 Stretching the processing volume (*inner box*) results in *S*(*j*)

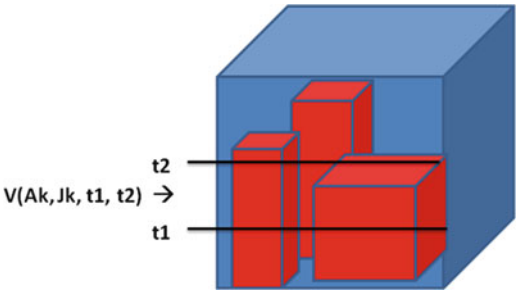


Fig. 1.3 A depiction of $V(A_k, J_k, t_1, t_2)$ as the sum of inner box volumes between $t1$ and $t2$

processed without accruing tardiness which provides a mitigating effect. Garcia and Rabadi (2013) combined these two ideas into a metric called the "stretched space consumption" $S(j)$ which balances a job's 2D space with its slack time. This metric is used to order jobs for area assignment in a manner analogous to largest-rectangle-first.

A job j may be thought of as a 3D box based on its width w_j, height h_j, and processing time p_j. This results in a "processing volume" $w_j h_j p_j$ in 3D space-time. A job may be processed at any time between its release time r_j and due date d_j without accruing any tardiness. If processing volume is held constant and stretched over the time interval $[r_j, d_j]$, this results in a new 2D space consumption $S(j)$ as shown in Fig. 1.2 (taken from Garcia and Rabadi 2013). Thus, $S(j)$ balances the job's 2D space requirement with its slack time, and is used to sequence jobs for area assignment. $S(j)$ is calculated as follows:

$$S(j) = \frac{p_j}{d_j - r_j} w_j h_j \tag{1.1}$$

Within any time interval $[t_1, t_2]$, the total available volume for job processing inside an area A_k with width W_k and height H_k is $(t_2 - t_1)\, W_k H_k$. In determining which area to assign to a job, $S(j)$ is used to estimate space consumption. Given a set of jobs J_k assigned to area A_k, $V(A_k, J_k, t_1, t_2)$ denotes the total estimated space consumption over the time interval $[t_1, t_2]$ based on the $S(j)$ metric. This is illustrated in Fig. 1.3 (taken from Garcia and Rabadi 2013), where the inner boxes represent the "stretched" jobs over their available processing time $[r_j, d_j]$. Here, $V(A_k, J_k, t_1, t_2)$ is the sum of inner box volumes that intersect $[t_1, t_2]$.

To calculate $V(A_k, J_k, t_1, t_2)$ it is necessary to identify the set of jobs J that intersect with $[t_1, t_2]$ as determined by their release times and due dates. Assuming stretched space consumption, in any interval $[t_1, t_2]$ the requested space $RS_j(t_1, t_2)$ is defined as follows:

$$\mathrm{RS}_j\,(t_1,\ t_2) = S(j)\left|[t_1,\ t_2] \cap \left[r_j,\ d_j\right]\right| \tag{1.2}$$

In (1.2), $\left|[t_1,\ t_2] \cap \left[r_j,\ d_j\right]\right|$ is used as shorthand to denote the length of the two interval intersections. $V(A_k, J_k, t_1, t_2)$ is then calculated follows:

$$V\,(A_k, J_k, t_1,\ t_2) = \sum_{j \in J_k} RS_j\,(t_1,\ t_2) \tag{1.3}$$

Thus, $V(A_k, J_k, t_1, t_2)$ is the total space consumption estimated during the time interval $[t_1, t_2]$ obtained by assuming each job j consumes $S(j)$ two-dimensional space over the time interval $[r_{jj}, d_j]$. The *packing factor* $P(A_k, J_k, t_1, t_2)$ is then defined as the fraction of estimated space consumption for area A_k during the time interval $[t_1, t_2]$:

$$P\,(A_k, J_k, t_1,\ t_2) = \frac{V\,(A_k, J_k, t_1,\ t_2)}{(t_2 - t_1)\ W_k H_k} \tag{1.4}$$

The area assignment heuristic first sorts jobs in largest- $S(j)$ -first order (analogous to sorting rectangles in largest-first order for a 2D packing problem). The heuristic uses a list of jobs assigned to each area, and jobs are assigned to areas one at a time in largest- $S(j)$ -first order. To assign each job j its processing area, j is first added to all the areas' assigned job lists. The packing factor is then calculated over the time interval $[r_j, d_j]$. The area with the smallest packing factor over this time interval retains j in its assigned job list, and j is deleted from the assigned job lists of all other areas. As a result, this area assignment method is called PF as shown in Algorithm 2. In the PF algorithm, a job fits inside of an area if its larger and smaller dimensions are not greater than the area's larger and smaller dimensions, respectively.

1.4.2 Job Input Orderings

After areas are assigned to jobs using PF, BLTI is used to produce schedules for each area separately. BLTI schedules jobs in the order in which they are given and accordingly, the schedule quality produced is entirely dependent upon job input ordering. Two separate ordering rules are used for inputting into BLTI: Earliest-Due-Date-First (EDD) and Max-Slack. EDD is a simple and well-known heuristic for machine scheduling problems with the objective of minimizing total tardiness. Max-Slack is an ordering specifically devised for this problem. Max-Slack sorts

Algorithm 2: PF with inputs *Areas* and *Jobs*

Procedure PF ($Areas = 1 \ldots m$, $Jobs = 1 \ldots n$) DO:
  Sort *Jobs* in Largest-First order based on $S(j)$ metric
  $J_k \leftarrow \emptyset$ for $k = 1 \ldots m$ // Job-area assignments initialized to empty set
  FOR $j \leftarrow 1$, $j \leq n$:
   $Best \leftarrow 1$ // Initialize to first area
   FOR $k \leftarrow 2$, $k \leq m$:
    IF $P\left(k, J_k \cup \{Jobs[j]\}, r_j, d_j\right) < P\left(Best, J_{Best} \cup \{Jobs[j]\}, r_j, d_j\right)$
    AND $Jobs[j]$ Fits Inside area k:
     $Best \leftarrow k$ // This is the best area found so far
  $J_{Best} \leftarrow J_{Best} \cup \{Jobs[j]\}$ // Assign Job j to Best Area

jobs on non-decreasing values of $d_j + r_j$, with the effect of giving priority to jobs with smaller release times and due dates. For each area, separate schedules are constructed using both EDD and Max-Slack and the better of the two is retained. These area schedules are then concatenated into a single final master schedule.

1.5 Local Search

In Garcia (2010) a number of heuristic and metaheuristic algorithms were developed for this spatial scheduling problem, including local search and several variants of simulated annealing. Extensive computational studies which compared the performance of the different algorithms were performed on a large set of benchmark problems covering a wide range of problem characteristics. Interestingly, local search (LS) was found to outperform the more sophisticated metaheuristic algorithms in almost all cases. Conceptually, LS works by keeping track of a best-so-far solution. A mutation operation is iteratively applied which randomly transforms one solution into another, and whenever a better solution is encountered the best-so-far is simply updated. LS was found to be a robust approach for spatial scheduling, and we present it in this section.

The LS algorithm utilizes BLTI to construct schedules for individual processing areas. As previously mentioned, the schedule quality generated by BLTI is entirely dependent upon the set of jobs assigned to the area and the input ordering of these jobs. Accordingly, LS performs a search by randomly mutating two attributes during each iteration: (1) job area assignment and (2) job input ordering. A mutation function applies these two operations to transform one solution into another, facilitating the search. Two parameters are used correspondingly to control the changing of job area assignments and job input ordering: *Area Change Probability* and *Swap Range*. The *Area Change Probability* is the probability that a given job is shifted to a new area. A master list of all jobs is kept by the LS algorithm which is used to feed jobs into BLTI in specified order. To alter the input ordering, two jobs

are randomly selected and their positions are swapped in the master list. The *Swap Range* limits the maximum number of positions apart between two jobs selected to swap positions. The mutate function is shown in Algorithm 3, and LS is shown in Algorithm 4.

Algorithm 3: Mutate operator with input *Job Sequence*

```
Procedure Mutate (Job Sequence):
  FOR job in Job Sequence :
    Randomly change area assignment if Uniform Rand in [0,1] ≤Area Change Probability
  (i, j) ← Random positions in Job Sequence at most Swap Range apart
  Swap jobs in positions i and j
```

Algorithm 4: Local Search with input *Jobs*

```
Procedure LS (Jobs):
  Best← Randomly order the elements in Jobs
  Randomly assign a processing area to each job in Best
  REPEAT Max Iteration TIMES:
    Next← Mutate(Best)
    Best←Next IF ObjectiveFunction(Next) is better than ObjectiveFunction(Best)
  RETURN Best
```

In Algorithm 4, BLTI is applied when evaluating the objective function of a job sequence. To evaluate the objective function of a job sequence the jobs are first separated into subsets by assigned processing area while retaining their relative sequence, then BLTI is applied for each subset, and finally the resulting schedules for each subset are then concatenated into a single master schedule. The objective function is then evaluated on the master schedule.

1.6 Metaheuristic for Randomized Priority Search (Meta-RaPS)

The Metaheuristic for Randomized Priority Search (Meta-RaPS) is a search control strategy that uses both construction and improvement heuristics to generate high quality solutions. Meta-RaPS has shown excellent results for many combinatorial optimization problems including the Resource Constrained Project Scheduling Problem (DePuy and Whitehouse 2001), the Traveling Salesman Problem (DePuy et al. 2005), the 0–1 Multidimensional Knapsack Problem (Moraga et al. 2005), the Vehicle Routing Problem (Moraga 2002), and the Parallel Machine Scheduling

Problem with Setup Times (Rabadi et al. 2006). More recently, Meta-RaPS was applied to the spatial scheduling problem (Garcia and Rabadi 2011), and this algorithm is presented in this section.

Meta-RaPS, like other metaheuristics, utilizes randomness as a mechanism to prevent becoming trapped into local optima. Meta-RaPS is controlled by four basic parameters: the number of iterations (I), the priority percentage ($p\%$), the restriction percentage ($r\%$), and the improvement percentage ($i\%$). During execution, the number of iterations determines the number of feasible solutions constructed. Each iteration consists of two stages: (1) a solution construction stage and (2) an (optional) improvement stage. During the construction stage a heuristic builds a new solution by systematically adding available jobs to the current schedule based on a priority rule. Rather than strictly adhering to the priority rule, however, Meta-RaPS deviates from this a certain percentage of the time to introduce diversity. The $p\%$ parameter determines the percentage of time the next item is added to the solution according to the priority rule. In the remaining time ($100\% - p\%$), the next item added to the solution is randomly chosen from a candidate list that consists of feasible items whose priority values are within the best $r\%$ of the best priority value. This mechanism allows the algorithm to avoid becoming trapped in a local optimum.

In the optional improvement stage, a solution improvement algorithm is included and the $i\%$ parameter is used to control this process. This is typically a local search or similar heuristic. The improvement heuristic is performed if the objective function value of the solution most recently constructed is within the best (lowest in this case) $i\%$ of the range between both best and worst unimproved solutions found so far. By only improving the best $i\%$, time is not wasted on unpromising solutions. Meta-RaPS continues iterating between these two stages until the maximum number of iterations is reached, and the best solution encountered over all iterations is reported.

1.6.1 Construction Stage

In the construction stage a complete schedule is constructed from an empty solution based on a heuristic rule. The heuristic rule used for this algorithm is to first sort jobs into EDD order, and then place each job in the processing area resulting in the earliest completion time. Jobs are placed in areas by using BLTI and diversification is introduced by selecting a random element from a candidate list ($1 - p\%$) of the time (instead of strictly adhering to the EDD ordering). The candidate list is obtained by selecting all jobs whose end time is no greater than $r\%$ percent of the difference between the minimum and maximum ending times of the remaining jobs. An implicit job ordering is obtained by selecting all the jobs with this combination of priority-based and randomized selection. This new job ordering is returned at the end of the construction stage and is used in the improvement stage. The constructive heuristic and candidate selection methods are described in Algorithm 5.

1.6.2 Improvement Stage

In this spatial scheduling algorithm we use a local search procedure to improve promising solutions. This procedure is a slightly modified version of the local search in the previous section (specifically Algorithm 4). The key difference is that there is no random initial solution; the jobs begin with the areas assigned by the construction procedure and in the same sequence as generated by the construction. The improvement procedure is shown in Algorithm 6.

Algorithm 5: Construct with input *Jobs, %p, and %r*

```
Procedure Construct (Jobs, %p, %r):
  Sort Jobs into EDD order
  New Job Order← []
  WHILE Jobs is not empty :
    IF uniform random in [0,1] > %p: j← RandomCandidate(Jobs, %r)
    ELSE: j ← Next(Jobs)
    Remove j from Jobs and add j to New Job Order
    Best Area← NIL
    Best Time← Infinity
    FOR each area A where j fits inside A :
      BLTI([j], A)
      IF s_j + p_j <Best Time:
        Best Area←A
        Best Time←s_j + p_j
    FOR each area A where A≠Best Area :
      Undo BLTI([j], A)
  Return New Job Order
// Auxiliary procedure used above
Procedure RandomCandidate(Jobs, %r):
  Min End← Min {s_j + p_j for j in Jobs}
  Max End← Max {s_j + p_j for j in Jobs}
  List← {j in Jobs where s_j + p_j≤Min End + [(Max End − Min End) ×%r)]}
  Return randomly selected element from List
```

Algorithm 6: Improve with input *Jobs*

```
Procedure Improve (Jobs):
  Best←Jobs
  REPEAT Max Improvement Iteration TIMES:
    Next← Mutate(Best)
    Best←Next IF ObjectiveFunction(Next) is better than ObjectiveFunction(Best)
  RETURN Best
```

1.6.3 Complete Meta-RaPS Algorithm

The main algorithm applies Meta-RaPS to this spatial scheduling problem by employing the construction and improvement heuristics as described above. As previously discussed, Meta-RaPS uses four basic parameters: the priority percent ($p\%$) determines the percentage of time in the construction stage that a randomly selected element from a candidate list is used versus the pure heuristic rule followed; the restriction percent ($r\%$) specifies the range of priorities that the candidate list may contain as a fraction of the difference between the least and greatest priorities; the number of iterations (I) specifies the number of solutions to be constructed and possibly improved; and finally the percent improvement ($i\%$) specifies a limit on how good a solution must be for it to be considered for improvement. The main Meta-RaPS spatial scheduling algorithm is shown in Algorithm 7.

Algorithm 7: Meta-RaPS-Schedule with input *Jobs, %p, %r, I,* and *%i*

Procedure MetaRaPS-Schedule(*Jobs, %p, %r, I, %i*):
  Best← NIL; *Best Constructed*← NIL; *Worst Constructed*← NIL
  REPEAT *I* times:
  Current← Construct(*Jobs, %p, %r*)
  IF in first iteration:
    Best Constructed←*Current*
    Worst Constructed←*Current*
  IF ObjectiveFunction(*Current*) < ObjectiveFunction(*Best Constructed*):
    *Best Constructe*d ←*Current*
  IF ObjectiveFunction(*Current*) > ObjectiveFunction(*Worst Constructed*):
    Worst Constructed←*Current*
  Range← ObjectiveFunction(*Best Constructed*) –
        ObjectiveFunction(*Worst Constructed*)
  IF ObjectiveFunction(*Current*) ≤ ObjectiveFunction(*Best Constructed*) + (*Range*×*%i*):
    Current← Improve(*Current*)
  IF ObjectiveFunction(*Current*) < ObjectiveFunction(*Best*):
    Best←*Current*
  Return *Best*

1.7 Comparison of Methods: Computational Results on Benchmark Problems

We compare the different algorithms discussed in this section to one another and also to the IP using the CPLEX solver. In this undertaking, the benchmark problems and experiment design from Garcia and Rabadi (2011) are used. Problems in this set

were generated by using three design factors: Average Slack Time, Release Rate, and Average Space Consumption. A job's slack time may be thought of as the difference between its processing time and the amount of time between its release time and due date. A job with less slack time will have less flexibility as to when it may be scheduled without accruing tardiness and consequently, a lower average slack time value will increase the problem difficulty. The Release Rate is the average number of jobs whose release times occur in a given time unit. A higher Release Rate will result in more jobs becoming available for processing at any given time and will likewise result in a more complex problem. Finally, a job's Space Consumption is the amount of 2D space required. A higher Average Space Consumption implies fewer jobs on average may be placed inside the processing areas at a time which also results in a more difficult problem. In this set of benchmark problems, each design factor has two levels, low (−) and high (+), resulting in eight different problem characteristic combinations. Thus, the full factorial experiment follows a 2^3 design. Each combination contains ten problem instances, resulting in a total of 80 problems. Each problem consists of 50 jobs. A more detailed description of the problem generation method is found below and all data are available at www.SchedulingResearch.Com.

1.7.1 Problem Generation

Test problems were generated using the Average Slack Time, Release Rate, and Average Space Consumption as discussed above. Each problem utilized the same processing area configuration consisting of four processing areas with width × height dimensions as follows: 30×23; 17×14; 12×10; and 15×15. The problem generation parameters along with their descriptions are shown in Table 1.1.

Uniform distributions are commonly employed in the generation of scheduling benchmark problems because they result in large levels of variance and thus subject solution methods to the most unfavorable conditions (Rabadi et al. 2006). For these reasons uniform distributions were utilized. The number of jobs used in all problems was $n = 50$ and the average job processing time was held constant at $\mu_p = 20$. The levels used for each of the design factor parameter are as follows: $\mu_a = 0.05$ (−) and 0.12 (+); $\mu_s = 0.5$ (−) and 0.12 (+); and finally $\lambda = 0.05$ (−) and 0.12 (+).

1.7.2 Parameter Settings and Algorithm Implementations

All algorithms were implemented in the Ruby programming language. Additionally, the IP model for this spatial scheduling problem was implemented using IBM ILOG OPL and solved using CPLEX 12.2 on a computer having a 2.5 GHz quad-core processor with 6 GB of memory. A 20-min time limit was imposed on CPLEX and in cases where the time limit was reached, the incumbent solution was reported.

Table 1.1 Problem generation parameters

Parameter	Distribution	Description
n	N/A	Number of jobs
μ_a	Uniform over $[\mu_a - 0.4\mu_a,\ \ \mu_a + 0.4\mu_a]$	Average job size fraction. This is a number in (0, 1) that denotes the fraction of total available processing space taken by the average job. For instance, if $\mu_a = 0.1$ this means that each job on average takes 10 % of total available processing space
μ_s	Uniform over $[\mu_s - 0.4\mu_s,\ \ \mu_s + 0.4\mu_s]$	Average slack multiple. Slack time is generated by a random number r with mean μ_s. The average slack multiple μ_s is a multiple of a job's processing time p_j, and the total amount of slack time for job j is $p_j r$
μ_p	Uniform over $\left[\mu_p - 0.4\mu_{p,}\ \ \mu_p + 0.4\mu_p\right]$	The average job processing time (or duration)
λ	N/A	The release rate

For the LS algorithm, the following parameters were used: *Swap Range* = 3; *Area Change Probability* = 0.05; and *Max Iterations* = 600. A time limit of 4 min was imposed on the LS algorithm, and it was also set to terminate if it exceeded 40 consecutive iterations with no improvement. Additionally, two runs per problem were performed and the best result of the two was reported in each case. LS was run on a 2.5 GHz dual-core processor with 7.5 GB of memory.

For MetaRaPS, the following parameters were used: $p\% = 10$ %; $r\% = 10$ %, $i\% = 10$ %; $I = 600$; *Area Change Probability* = 0.05; *Swap Range* = 2; and $LSI = 5$. It is pointed out that the maximum number of iterations I is the same for Meta-RaPS as for LS above, in order to allow proper comparison between the two methods. Similarly, a time limit of 4 min was imposed on the Meta-RaPS algorithm and it was also set to terminate if it exceeded 40 consecutive iterations with no improvement. Two runs per problem were performed and the best result of the two was reported in each case. Meta-RaPS was run on a 2.4 GHz quad-core computer with 2 GB of memory.

The PF algorithm was run on a computer having a 2.5 GHz quad-core processor with 6 GB of memory.

1.7.3 Results and Discussion

The results of the comparative experiments (in terms of average objective function and solution time) are shown in Table 1.2. It is seen that PFH requires very small amounts of processing time to in many cases produce solutions nearly as good as LS. The best solutions were found by CPLEX and Meta-RaPS in the vast majority of cases. Meta-RaPS produced the best solutions in most problem classes where

Table 1.2 Summary of experiment results

Combination			Avg. objective value				Avg. solution time (s)			
μ_s	λ	μ_a	CPLEX	PFH	LS	Meta-RaPS	CPLEX	PFH	LS	Meta-RaPS
−	−	−	0	15.13	5.09	8.13	106.55	1.92	57.3	209.08
−	−	+	1338.1	2484.76	2053.61	1768.19	1199.73	2.64	57.58	183.24
−	+	−	389.51	410.25	378.95	306.31	1114.43	1.75	71.65	184.65
−	+	+	2973.7	3085.17	3078.89	2754.46	1199.74	2.27	61.92	218.91
+	−	−	0	0.88	0	0	43.62	1.89	5.64	66.54
+	−	+	541.12	1261.22	1077.41	943.92	1199.66	2.5	67.4	207.96
+	+	−	1.87	3.99	0.35	0.43	693.25	1.54	22.93	75.77
+	+	+	1270.2	1805.81	1808.28	1519.38	1199.76	2.1	66.27	195.95

CPLEX frequently reached the time limit (e.g., >1110 s average time). In one problem class (+ + −) LS trivially produced the best solutions on average.

All benchmark problems in this set consisted of 50 jobs. As clearly seen, CPLEX took far more time in most cases than all the other methods and reached the time limit in the majority of the problems in this set. This suggests that problems of this size approach the upper limit of cases where IP can be used. Meta-RaPS in many cases beat CPLEX in much smaller amounts of time and seldom significantly underperformed it. Meta-RaPS appears to be a robust method for solving spatial scheduling problems. Solutions to all instances are available at www.SchedulingResearch.com.

1.8 Conclusion

In this chapter we have examined a general type of spatial scheduling problem involving multiple non-identical processing areas used in parallel, with the objective of minimizing total tardy time. Three approximation algorithms have been shown for this problem: a non-iterative method PFH, local search (LS), and Meta-RaPS. These three methods were compared to one another and to IP in using a set of benchmark problems in a computational experiment. PF produced many good solutions in a very short amount of time. Meta-RaPS showed consistently strong performance and beat IP in many cases on more difficult problems.

References

Baker BS, Coffman EG, Rivest RL (1980) Orthogonal packings in two dimensions. SIAM J Comput 9:846–855

Caprace J-D, Petcu C, Velarde G, Rigo M (2013) Optimization of shipyard space allocation and scheduling using a heuristic algorithm. J Mar Sci Technol 18(3):404–417

Cho KK, Chung KH, Park C, Park JC, Kim HS (2001) A spatial scheduling system for block painting process in shipbuilding. CIRP Ann Manuf Technol 50:339–342

DePuy G, Whitehouse GE (2001) A simple and effective heuristic for the multiple resource allocation problem. Int J Prod Res 32(4):24–31

DePuy GW, Moraga RJ, Whitehouse GE (2005) Meta-RaPS: a simple and effective approach for solving the traveling salesman problem. Transp Res E: Logist Transp Rev 41(2):115–130

Duin CW, Van Sluis E (2006) On the complexity of adjacent resource scheduling. J Sched 9:49–62

Garcia C (2010) Optimization models and algorithms for spatial scheduling. Ph.D. Thesis, Old Dominion University, Norfolk

Garcia C, Rabadi G (2011) A meta-RaPS algorithm for spatial scheduling with release times. Int J Plann Sched 1(1/2):19–31

Garcia C, Rabadi G (2013) Exact and approximate methods for parallel multiple-area spatial scheduling with release times. OR Spectr 35(3):639–657

Hopper E, Torton BCH (2001) An empirical investigation on metaheuristic and heuristic algorithms for a 2D packing problem. Eur J Oper Res 128:34–57

Lee KJ, Lee JK (1996) A spatial scheduling system and its application to shipbuilding: DAS-curve. Expert Syst Appl 10:311–324

Moraga RJ (2002) Meta-RaPS: an effective solution approach for combinatorial problems. Ph.D. thesis, University of Central Florida, Orlando

Moraga RJ, DePuy GW, Whitehouse GE (2005) Meta-RaPS approach for the 0–1 multidimensional knapsack problem. Comput Ind Eng 48(2):83–96

Padberg M (2000) Packing small boxes into a big box. Math Meth Oper Res 52:1–21

Paulus JJ, Hurink J (2006) Adjacent resource scheduling: why spatial resources are so hard to incorporate. Electron Notes Discrete Math 25:113–116

Rabadi G, Moraga R, Al-Salem A (2006) Heuristics for the unrelated parallel machine scheduling problem with setup times. J Intell Manuf 17:85–97

Srinivasan S (2014) Spatial scheduling algorithms for production planning problems. Ph.D. Thesis, Virginia Commonwealth University, Richmond

Varghese R, Yoon, DY (2005) Spatial block arrangement in shipbuilding industry using genetic algorithm for obtaining solution for anticipated bottleneck. Proceedings of the Fifteenth International Offshore and Polar Engineering Conference, vol 4, pp 774–780

Chapter 2
Estimating the Costs of Planned Changes Implied by Freezing Production Plans

Po-Chen Lin and Reha Uzsoy

Abstract The use of production planning algorithms on a rolling horizon basis is very common in practice. However, this leads to frequent changes in planned quantities for future periods which may adversely impact support activities such as material preparation, staffing, and setup planning. In this chapter we examine two widely used approaches for this problem, the use of change costs to penalize changes in planned quantities and freezing of the plan by prohibiting any changes in some number of periods in the near future. We use a linear programming model of a single-product single-stage system to develop insights into the conditions when the two approaches are equivalent. Specifically, we derive lower bounds on the values of the change costs which will ensure freezing of the plan in a given planning epoch, and present numerical results to illustrate our findings.

Keywords Rolling horizon • Production planning • Nervousness • Change costs • Freezing

2.1 Introduction

Rolling horizon procedures, where an infinite horizon problem is approximated by the solution to a sequence of finite horizon problems, are common in production planning practice and research. We define the points in time at which a finite horizon model is solved as a planning epoch s, and the interval of time consisting of the T periods $t = s, s+1, \ldots, s+T-1$ covered by its planned decisions as the planning window T. Thus at each epoch new values of key decision variables, especially planned release quantities, are calculated, resulting in changes in the planned values of these variables from one epoch to the next. The changes in planned release quantities that occur in each period as new updated demand information becomes

P.-C. Lin • R. Uzsoy (✉)
Edward P. Fitts Department of Industrial and Systems Engineering, 300 Daniels Hall, North Carolina State University, Raleigh, NC 27695-7906, USA
e-mail: plin4@ncsu.edu; ruzsoy@ncsu.edu

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_2

available may disrupt supporting activities such as staffing, material procurement, and machine setup that are initiated based on plans developed in earlier periods. This phenomenon, referred to in the literature as *schedule nervousness* or *stability*, has been addressed by many researchers over the last several decades (Mather 1977; Carlson et al. 1979; Blackburn et al. 1985, 1986; Sridharan et al. 1988; Narayanan and Robinson 2010).

Several approaches have been proposed to alleviate scheduling nervousness, such as introducing ending conditions in the model solved at each epoch (Eilon 1975; Hung and Leachman 1996; Fisher et al. 2001; Voss and Woodruff 2006), forecasting beyond the current planning horizon (Grinold 1980; Carlson et al. 1982; Grinold 1983; Kropp et al. 1983; Blackburn et al. 1985, 1986), holding safety stock (Kropp et al. 1983; Blackburn et al. 1985, 1986; Yano and Carlson 1987; Sridharan and LaForge 1989; Metters and Vargas 1999; Bai et al. 2002; Sahin et al. 2013), freezing the schedule by prohibiting changes in certain periods (Blackburn et al. 1985, 1986; Sridharan et al. 1987, 1988; Sridharan and Berry 1990; Zhao and Lee 1993; Zhao and Xie 1998; Zhao and Lam 1997; Xie et al. 2003), and introducing change costs (Carlson et al. 1979; Kropp et al. 1983; Voss and Woodruff 2006) and chance constraints (Johnson and Montgomery 1974; Bookbinder and Tan 1988; Tarim and Kingsman 2004; Ravindran and Uzsoy 2011; Aouam and Uzsoy 2014). In this chapter we focus on the link between two prominent methods: freezing the schedule by prohibiting planned changes for some subset of the periods in the current planning window, and introducing change costs in the objective function that penalize planned changes, in a single-stage single-product system with fixed lead time and limited capacity. We assume the planning problem solved at each epoch of the rolling horizon procedure takes the form of a linear program, and analyze the structure of optimal solutions to develop lower bounds on the values of change costs that will guarantee zero planned changes in a given period of the current epoch, i.e., freezing the schedule, at optimality. We thus demonstrate the equivalence between freezing the schedule and penalizing planned changes in the objective function.

In the next Sect. 2.2, we will briefly discuss the methods of freezing schedules and introducing change costs. In Sect. 2.3, we present the primal and dual formulations for a single-stage single-product production system. In Sect. 2.4, we analyze the behavior of release changes for a single-product model. Detailed analysis of unit release change costs to guarantee freezing is provided in Sect. 2.5. A computational study illustrates the implications of our findings in Sect. 2.6. Section 2.7 summarizes our conclusions and discusses future research directions.

2.2 Literature Review

In this section, we briefly introduce the rolling horizon approach used in this chapter, and then discuss the methods of freezing the schedule and introducing change costs.

2.2.1 Rolling Horizon Approach

In the most common application of the rolling horizon approach, time is considered in discrete periods $t = 1, \ldots, \infty$. At the start of each planning epoch s, all relevant information on the state of the production system and estimated future demand is collected and a production plan developed for the next T periods $s, s+1, \ldots, s+T-1$ using a finite horizon planning model whose nature may vary with the application domain. The specific decisions usually involve planned material release quantities $R_t(s)$ and production quantities $X_t(s)$ for each period as well as lot sizes in problems involving setup times. The decisions for the current period s are implemented, and time advances to the beginning of period $s + 1$, when the process of information collection and planning recommences. We shall refer to each period s at which a finite horizon plan is developed as a *planning epoch*, the set of periods s, $s + 1, \ldots,$ $s + T - 1$, as the current *planning window*, and the number of periods T considered in the finite horizon model as the *planning window length*. We shall denote the set of all information available for planning purposes at the start of period s as $\Omega(s)$ and the solution obtained by the planning model solved at the start of period s as $\mathbf{R(s)} = [R_s(s), R_{s+1}(s), \ldots, R_{s+T-1}(s)]$ where $R_t(s)$, $\forall t = 1, \ldots, K$ denotes the values of the decision variables $R_t(s)$ computed for period t at planning epoch s, $s \leq t \leq s + T - 1$. K is the final period of the entire planning horizon.

A natural consequence of this process is that the decision variables $R_t(s)$ associated with a given period t, $t \leq s$ and $0 \leq s-t \leq T-1$, are revised $T-1$ times before actually being implemented in period $t = s$. The changes arise from the fact that the finite horizon planning models used to develop plans at successive planning epochs use different sets of information, i.e., $\Omega(s) \subset \Omega(s + 1)$. A given period t, $t \leq s$ and $0 \leq s-t \leq T-1$, will first be considered at planning epoch $s = t-T+1$, yielding decision variable $R_t(t - T + 1)$. The next planning epoch will yield a new set of decision variables $R_t(t - T + 2)$ for period t. Eventually, after new decisions $R_t(t - T + 1), R_t(t - T + 2), \ldots, R_t(t - 1)$ have been calculated at planning epochs $s = t - T + 1, \ldots, t - 1$, the decision variable $R_t(s)$ will be implemented at epoch $s = t$. The basic rolling horizon procedure we study is thus described as follows:

2.2.2 Algorithm RH

Step 1: Set $s = 0$, given the initial WIP level $W_0(s)$ and finished goods inventory (FGI) levels $I_0(s)$.

Step 2: Increment s by 1.

Step 3: Solve the finite horizon problem with planning window length T involving periods s to $s + T - 1$.

Step 4: Implement the releases $R_s(s)$ and production $X_s(s)$ as the realized plan. Record the WIP $W_s(s)$ and FGI $I_s(s)$ at the end of period s as initial values of

WIP $W_s(s) = W_s(s+1)$ and FGI $I_s(s) = I_s(s+1)$ for the next window from periods $s+1$ to $s+T$.

Step 5: Update the demand forecasts for periods $s+1$ to $s+T$.

Step 6: Return to Step 2 and repeat until epoch $s = K - T + 1$.

This repeated revision of the planning decisions for each period can cause problems in practice. One of the purposes of production planning is to provide visibility into future requirements for ancillary functions that support production, such as staffing, maintenance, and purchasing. Frequent changes in planned release and production quantities can lead to significant disruptions in these support activities, often rendering the execution of the production plans infeasible and leading to much wasted effort due to redundant updates. In the notation above, the change in the planned release quantities $R_t(s)$ and $R_t(s+1)$ from epoch s to epoch $s+1$ is given by:

$$\triangle R_t(s, s+1) = R_t(s+1) - R_t(s).$$

We shall refer to these changes as *planned changes* in the production plan from epoch s to the next epoch $s+1$. In this study, we focus on planned changes in only one set of decision variables, the release variables $R_t(s)$, because these decisions affect both production planning and its supporting activities.

2.2.3 Freezing the Schedule Within the Planning Horizon

In many production environments it is important to maintain a stable production plan where decisions for a given time period are not changed dramatically from epoch to epoch. Frequent, large changes in previously announced plans will cause the users of those plans to question their reliability; if they feel plans are unreliable, users will tend to ignore any planned releases and production that they think are likely to change in the future. Supporting activities such as staffing, machine preparation, and materials procurement need time to respond to changes in planned releases, which may cause longer cycle times for the entire production system. Therefore, a stable plan is preferable.

Freezing the schedule fixes the production plans for the next several periods, permitting no changes even when there are changes in demand. This has the advantage of eliminating planned changes in the frozen periods, but also limits the system's ability to respond to demand changes. When demand forecasts are revised upwards, freezing may result in unmet demand and reduced customer service levels. When demand forecasts are revised down in the face of reduced demand, excess inventories may accumulate. Plans associated with periods further in the future are allowed to change.

Freezing the schedule was proposed with the aim of controlling schedule nervousness (Sahin et al. 2013). This strategy aims to divide a single planning

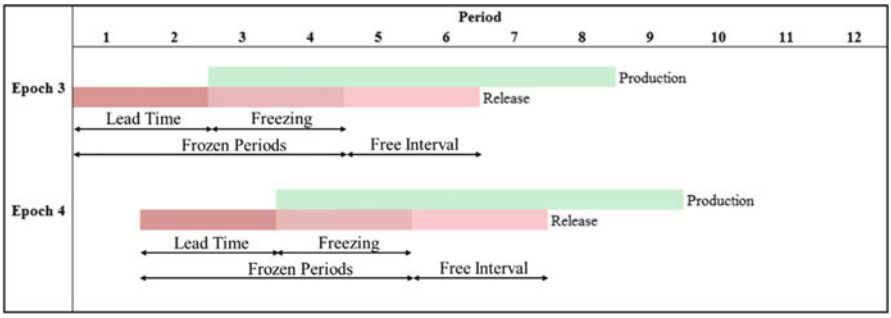


Fig. 2.1 Concept of freezing the schedule

window into two parts as shown in Fig. 2.1: frozen periods and a free interval. The frozen periods, in turn, consist of two parts: lead time and additional freezing periods. In a rolling horizon environment, the output for the first L periods is determined by releases implemented in periods prior to the current planning window and cannot be changed. Freezing a period implies a decision not to change the planned release quantities even when demand has changed. We can only change releases in periods in the free interval in response to demand changes.

Many researchers (Blackburn et al. 1985, 1986; Sridharan et al. 1987, 1988; Sridharan and Berry 1990; Zhao and Lee 1993; Zhao and Xie 1998; Zhao and Lam 1997; Xie et al. 2003) have proposed freezing schedules to provide more stable production plans. There are two general freezing approaches, order-based and period-based freezing (Sridharan et al. 1987; Zhao and Lee 1993). Under order-based freezing we freeze specific orders, while period-based freezing freezes all orders in the schedule for specified periods. In this chapter we consider the latter approach. In terms of schedule stability, Zhao and Lee (1993) found that period-based freezing with longer freezing periods outperforms order-based freezing in terms of schedule changes, but with lower service level and higher costs. Sridharan et al. (1987) found that in an uncapacitated system freezing up to 50 % of the planning window increases production changeover and inventory carrying costs only slightly. Zhao and Lee (1993) further found that under deterministic demand, longer freezing periods affect cost performance, production instability, and service level only slightly. Sridharan and LaForge (1994a) also found that freezing part of the schedule reduces the service level slightly. However, under stochastic demand, longer freezing periods degrade all performance measures. They suggested balancing schedule stability and service level by choosing an appropriate length of the freezing period when demand is stochastic.

In order to increase the service level in the presence of frozen schedules, Yano and Carlson (1987) and Sridharan and LaForge (1989) suggested holding safety stock. In the presence of safety stock, Sridharan and LaForge (1989, 1994a,b) found that freezing a part of the schedule may reduce the service level only slightly. By specifying safety stock levels, Sridharan and Berry (1990) claimed

that freezing the schedules can reduce schedule instability under both deterministic and stochastic demand. Blackburn et al. (1985, 1986) presented a comprehensive comparison of four different production strategies to improve schedule stability: lot for lot, freezing the schedule, introducing change costs, and extending the planning window by forecasting using the Wagner-Whitin algorithm and Silver-Meal heuristic. They suggested that under most conditions, freezing the schedule and implementing change costs yield better cost and schedule stability than the other planning strategies. They also pointed out that when demand is uncertain with fixed capacity and lead time, freezing the schedule only helps schedule stability. The question of how to select proper values for freezing window length and change costs is left open. Thus, there is a tradeoff between schedule stability, on the one hand, and other performance measures such as service level and total cost, on the other.

2.2.4 Introducing Change Costs

The method of introducing change costs also aims to control schedule nervousness by penalizing both positive and negative planned changes from one planning epoch to the next (Voss and Woodruff 2006). Change costs represent a unit cost imposed on any planned change from the planned quantity computed in the previous planning epoch. Carlson et al. (1979) and Kropp et al. (1983) also proposed incorporating schedule change costs into material requirement planning (MRP) using the Wagner-Whitin (1958) algorithm in a rolling horizon environment with setup costs. They did not consider unit change costs but focused instead on the introduction of additional setups, without cancelling an existing setup. They suggested that with schedule change costs an optimization model will choose the most economical solution, but did not discuss how to set appropriate values for change costs. Kropp and Carlson (1984) introduced change costs for adding or cancelling a setup and suggested, following Carlson et al. (1979) and Kropp et al. (1983), that schedule change costs can help to control the tradeoff between holding and setup costs in order to minimize total overall costs. Based on the contributions of Carlson et al. (1979), Kropp et al. (1983) and Kropp and Carlson (1984), Blackburn et al. (1985, 1986) further modified the change costs to encourage new setups and discourage cancellation of an already planned setup. They compared the change cost method to other strategies such as lot sizing, freezing schedules, and holding safety stocks in a rolling horizon environment, finding that adding change costs and freezing the schedule provide better results. However, the problem of specifying appropriate values of the change costs is critical to the effective use of this approach. Setting change costs too high may result in increased costs due to excess inventories or backorders, while setting them too low may permit unnecessary changes in planned quantities.

Braun and Schwartz (2012) examine three different methods to reduce schedule nervousness in master production scheduling: (1) Freezing the schedule, (2) Move suppression, and (3) Schedule change suppression. The frozen horizon approach involves freezing the schedule for a certain number of periods in each planning

window. Move suppression introduces a penalty cost on the changes from period $t-1$ to the next period t in the same planning epoch. Schedule change suppression introduces unit change costs on any change made in any planned quantity from one epoch to the next. Both move suppression and schedule change suppression are controlled by a penalty factor trying to minimize the level of schedule nervousness, defined as the maximum amount of change from period to period. Under this objective, move suppression performs the best, followed by schedule change suppression. The frozen horizon approach creates the most nervousness because it needs to react to fulfill demand. The penalty cost plays a critical role in controlling both the timing and magnitude of schedule changes. However, determining a specific value for the penalty cost requires considerable trial and error. In addition, the three approaches do not consider capacity and holding cost, yielding no clues as to the impact of the policies on backlogs and inventory costs.

Cost structure is also a critical factor that affects the performance of planning systems. Several researchers (Sridharan et al. 1987; Sridharan and Berry 1990; Zhao and Lee 1993; Kadipasaoglu and Sridharan 1995) have found that the ratio of setup and FGI holding costs influences schedule stability, service level, and total cost performance in MRP. A large setup to holding cost ratio produces fewer setups, generating lower total cost and more stable plans. However, these findings assume unlimited capacity and no backlogging. Voss and Woodruff (2006) impose a common nonnegative change cost on both positive or negative changes. They also suggest that a quadratic penalty function is more realistic and simpler, removing the need to distinguish between positive and negative changes. However, this results in a nonlinear objective and does not consider the different causes of positive or negative changes. Thus we still lack clear insight into how to set change costs when planning in a rolling horizon environment with capacity constraints.

The approach of introducing release change costs in the objective function is equivalent to period-based freezing (Carlson et al. 1979; Kropp et al. 1983; Kropp and Carlson 1984; Blackburn et al. 1985, 1986; Lin et al. 1994; Voss and Woodruff 2006; Braun and Schwartz 2012) by setting the change costs to infinity for the frozen periods, and zero for the periods in the future where changes are permitted. However, this limiting case does not assist us in setting appropriate values for change costs. The costs of planned changes are hard to determine in practice, since they are driven by the changes to planned activities that must be made due to changes in production plans. Lin et al. (1994) set arbitrary unit change costs and found that the length of the frozen periods increases as the unit change cost increases. Hence it is useful to identify the minimum value of the change cost in a particular period that will ensure that the schedule will be frozen in that period. If an accurate estimate of this value could be computed, management could compare this estimate with their knowledge of the system and the potential impacts of planned changes to assess whether the decision to freeze the plan is justified.

In the next Sect. 2.3, we will begin our analysis of the release change cost LP model with fixed lead time to provide comprehensive analysis of how to set unit release change costs to guarantee freezing the schedule in the current epoch.

2.3 Release Change Cost Model with Fixed Capacity

In this section, we present the mathematical model of a single-product single-stage production system with a finite planning window.

2.3.1 Single-Product Model

In this section, we consider a single-stage single-item production system with a deterministic production lead time of L periods and a capacity of C units per planning period. We introduce unit release change costs to the model motivated by Voss and Woodruff (2006) using the following notation:

Indices:

s: planning epoch $s = 1, \ldots, K-T+1$; also indicates the first period of a planning epoch.
t: a planned period within the planning window from s to $s + T - 1$.

Parameters:

T: planning window length, the number of periods considered in a planning epoch consisting of periods s to $s + T - 1$.
K: number of planning periods in the entire horizon.
C: maximum number of units the system can produce in a planning period.
L: lead time-material released to the system in period t becomes available as finished product in period $t + L - 1$.
$D_t(s)$: demand forecast made at the start of epoch s for period t, $s \leq t \leq s + T - 1$.
ω_t: unit WIP holding cost.
φ_t: unit FGI holding cost.
π_t: unit backlog cost.
γ_t: unit release change cost.
$R_t^{'}(s-1)$: release quantity planned for period t in planning epoch $s - 1$, which is outside the current planning epoch s. These values correspond to planned quantities from the previous decision epoch $s - 1$, and hence are known with certainty at the start of the current planning epoch s.
$\overline{R_t(s)}$: release quantity already implemented for period $t = s-L+1, \ldots, s-1$. These are deterministic parameters that must be considered due to the presence of the fixed lead time L.
U: estimated demand for all periods outside the current planning window, corresponding to the estimated mean of future demand.

Primal Decision Variables:

$R_t(s)$:	release quantity planned in epoch s for period t, $s \leq t \leq s+T-1$.
$W_t(s)$:	WIP level planned in epoch s for period t, $s \leq t \leq s+T-1$.
$X_t(s)$:	production quantity planned in epoch s for period t, $s \leq t \leq s+T-1$.
$I_t(s)$:	FGI planned in epoch s for period t, $s \leq t \leq s+T-1$.
$B_t(s)$:	backlog planned in epoch s for period t, $s \leq t \leq s+T-1$.
$\triangle R_t(s-1,s)$:	planned release change for period t between the consecutive epochs $s-1$ and s , $\triangle R_t(s-1,s) = R_t(s) - R_t^{'}(s-1)$.

Using this notation we can formulate the single-product planning model solved at each epoch of the rolling horizon procedure as follows:

2.3.2 Mathematical Model

Objective:

$$\text{Minimize} = \left\{ \sum_{t=s}^{s+T-1} \left[\omega_t W_t(s) + \varphi_t I_t(s) + \pi_t B_t(s)\right] + \sum_{t=s}^{s+T-2} \gamma_t \left|\triangle R_t(s-1,s)\right| \right\}.$$

Constraints:

$$R_{s+T-\tau}(s) \geq U, \ \forall\, \tau \in (1,L).$$

$$W_t(s) = \sum_{\tau=0}^{L-1} R_{t-\tau}(s), \ \forall\, t \in (s, s+T-1).$$

$$I_t(s) - B_t(s) = I_{t-1}(s) - B_{t-1}(s) + X_t(s) - D_t(s), \ \forall\, t \in (s, s+T-1).$$

$$X_t(s) \leq C, \ \forall\, t \in (s, s+T-1).$$

$$X_t(s) = R_{t-L}(s), \ \forall\, t \in (s, s+T-1).$$

$$\triangle R_t(s-1,s) = R_t(s) - R_t^{'}(s-1), \ \forall\, t \in (s, s+T-1).$$

$$X_t(s),\ W_t(s),\ R_t(s),\ I_t(s),\ B_t(s) \geq 0, \ \forall\, t \in (s, s+T-1).$$

While the WIP costs are usually omitted in planning models with fixed lead times (Missbauer and Uzsoy 2011), we include them in this model because of the role of WIP holding costs in determining release change costs which will emerge from our analysis. In order to analyze the structure of optimal solutions for this model, we first rewrite the release change variable $\triangle R_t(s-1,s)$, which is free in sign, as the difference of two nonnegative decision variables representing the positive and negative release changes:

$$\begin{aligned}\triangle R_t(s-1,s) &= R_t(s) - R_t^{'}(s-1) \\ &= \triangle R_t^{+}(s-1,s) - \triangle R_t^{-}(s-1,s).\end{aligned}$$

A positive release change $\triangle R_t^{+}(s-1,s) = \max\left[R_t(s) - R_t^{'}(s-1), 0\right] > 0$ means that the period t release planned in epoch s exceeds that planned in epoch $s-1$, so it is necessary to increase the planned release quantity to account for this change. On the other hand, $\triangle R_t^{-}(s-1,s) = \max\left[R_t^{'}(s-1) - R_t(s), 0\right] > 0$ means it was previously planned to release more material in epoch $s-1$ than is now needed in epoch s. We assign positive and negative release changes unit costs of γ_t^{+} and γ_t^{-}, respectively, and rewrite variables $X_t(s)$, $I_t(s)$, and $R_t(s)$, in terms of $\triangle R_t^{+}(s-1,s)$ and $\triangle R_t^{-}(s-1,s)$. This straightforward but tedious substitution yields the following primal formulation:

2.3.3 Primal Model

$$\begin{aligned}\text{Minimize} = &\sum_{t=s}^{s+T-1}\left[\left(\varphi_t + \pi_t\right) B_t(s)\right] + \omega_{s+T-1} W_{s+T-1}(s) \\ &+ \sum_{t=s}^{s+T-2}\left\{\triangle R_t^{+}(s-1,s)\left[\gamma_t^{+} + \sum_{\tau=t}^{t+L-1}\omega_t + \sum_{\tau=t}^{s+T-1}\varphi_t\right]\right\} \\ &+ \sum_{t=s}^{s+T-2}\left\{\triangle R_t^{-}(s-1,s)\left[\gamma_t^{-} - \sum_{\tau=t}^{t+L-1}\omega_t - \sum_{\tau=t}^{s+T-1}\varphi_t\right]\right\}\end{aligned}$$

Primal Constraints:

$$R_{s+T-1}(s) \geq U, \qquad \left[\alpha_{s+T-1}(s)\right]. \tag{2.1}$$

$$\begin{aligned}&\triangle R_{s+T-t}^{+}(s-1,s) - \triangle R_{s+T-t}^{-}(s-1,s) \geq U - R_{s+T-t}^{'}(s-1) \\ &\qquad \forall\, t \in (2,L),\ s \geq 2,\ L \geq 1, \qquad \left[\alpha_{s+T-t}(s)\right].\end{aligned} \tag{2.2}$$

$$\begin{aligned}&\sum_{\tau=s+L}^{t}\left[\triangle R_{s+T-t}^{+}(s-1,s) - \triangle R_{s+T-t}^{-}(s-1,s)\right] + B_t(s) \geq V_t(s), \\ &\qquad \forall\, t \in (s, s+T-1), \qquad \left[\beta_t(s)\right].\end{aligned} \tag{2.3}$$

$$-\triangle R^{+}_{s+T-t}(s-1,s)+\triangle R^{-}_{s+T-t}(s-1,s)\geq -C+R^{'}_{t}(s-1),$$
$$\forall\, t\in(s,s+T-1-L),\qquad [\sigma_t(s)]\,. \tag{2.4}$$

$$\triangle R^{+}_{s+T-t}(s-1,s)-\triangle R^{-}_{s+T-t}(s-1,s)\geq -R^{'}_{s+T-t}(s-1),$$
$$\forall\, t\in(s,s+T-2),\qquad [\eta_t(s)]\,. \tag{2.5}$$

$$B_t(s)\geqslant 0. \tag{2.6}$$

$$R^{+}_{s+T-t}(s-1,s),\ R^{-}_{s+T-t}(s-1,s)\geq 0,\ \forall\, t\in(s,s+T-2). \tag{2.7}$$

where for brevity of notation we define the constants:

$$V_t(s)=-I_{s-1}(s)+B_{s-1}(s)+\sum_{\tau=s}^{t}D_\tau(s)-\sum_{\tau=s+L}^{t}R^{'}_{\tau-L}(s)-\sum_{\tau=s}^{s+L-1}R_{\tau-L}(s). \tag{2.8}$$

Constraint set (2.4) implies that release changes must be feasible with respect to the residual capacity given by $R^{'}_{t}(s-1)-C$ in each period t. The dual variables associated with each constraint set are denoted by the Greek letters in square brackets to the right of the constraints.

2.3.4 Dual Model

The dual of this model is as follows:

$$\begin{aligned}\text{Maximize} = {} & U\alpha_{s+T-1}(s)+\sum_{t=2}^{L}\alpha_{s+T-t}(s)\left[U-R^{'}_{s+T-t}(s-1)\right]\\ & +\sum_{t=s}^{s+L-1}\left\{\beta_t(s)\left[-I_{s-1}(s)+B_{s-1}(s)+\sum_{\tau=s}^{t}D_\tau(s)+\sum_{\tau=s}^{t}R_{\tau-L}(s)\right]\right\}\\ & +\sum_{t=s}^{s+T-1}V_t(s)\beta_t(s)+\sum_{t=s}^{s+T-1-L}\sigma_t(s)\left[-C+R^{'}_{t}(s-1)\right]\\ & -\sum_{t=s}^{s+T-2}R^{'}_{t}(s-1)\eta_t(s).\end{aligned}$$

Dual Constraints:

$$\alpha_{s+T-1}(s)\leq\omega_{s+T-1},\ \forall\, t\in(s,s+T-1),\qquad [R_{s+T-1}(s)]\,. \tag{2.9}$$

$$\beta_t(s) \leq \varphi_t + \pi_t, \ \forall\, t \in (s, s+T-1-L), \qquad [B_t(s)]\,. \tag{2.10}$$

$$\begin{aligned} -\sigma_t(s) + \eta_t(s) + \sum_{\tau=t+L}^{s+T-1} \beta_\tau(s) &\leq Q_t^+(s), \\ \forall\, t \in (s+T-1, s+T-2), &\quad \left[\triangle R^+_{s+T-t}(s-1,s)\right]. \end{aligned} \tag{2.11}$$

$$\begin{aligned} \eta_t(s) + \alpha_t(s) &\leq \gamma_t^+ + \sum_{\tau=t}^{t+L-1} \omega_\tau, \\ \forall\, t \in (s, s+T-1-L), &\quad \left[\triangle R^+_{s+T-t}(s-1,s)\right]. \end{aligned} \tag{2.12}$$

$$\begin{aligned} \sigma_t(s) - \eta_t(s) - \sum_{\tau=t+L}^{s+T-1} \beta_\tau(s) &\leq Q_t^-(s), \\ \forall\, t \in (s+T-1, s+T-2), &\quad \left[\triangle R^-_{s+T-t}(s-1,s)\right]. \end{aligned} \tag{2.13}$$

$$\begin{aligned} -\eta_t(s) - \alpha_t(s) &\leq \gamma_t^- - \sum_{\tau=t}^{t+L-1} \omega_\tau, \\ \forall\, t \in (s, s+T-1-L), &\quad \left[\triangle R^-_{s+T-t}(s-1,s)\right]. \end{aligned} \tag{2.14}$$

$$\alpha_t(s),\ \beta_t(s) \geq 0,\ \forall\, t \in (s, s+T-1). \tag{2.15}$$

$$\sigma_t(s),\ \eta_t(s) \geq 0,\ \forall\, t \in (s, s+T-2). \tag{2.16}$$

For constraint sets (2.12) and (2.14), we define the constants:

$$Q_t^+(s) = \gamma_t^- + \sum_{\tau=t}^{t+L-1} \omega_\tau + \sum_{\tau=t}^{t+L-1} \varphi_\tau,\ \forall\, t \in (s, s+T-1-L).$$

and

$$Q_t^-(s) = \gamma_t^- - \sum_{\tau=t}^{t+L-1} \omega_\tau - \sum_{\tau=t}^{t+L-1} \varphi_\tau,\ \forall\, t \in (s, s+T-1-L)$$

Before we start to analyze release changes, we will briefly discuss some of the primal and dual constraints. Let us begin from constraint (2.3). When

$$I_t(s) = B_t(s) + \sum_{\tau=s+L}^{t} \left[\triangle R_t^+(s-1,s) - \triangle R_t^-(s-1,s)\right] - V_t(s) > 0.$$

the system has positive FGI, implying $\beta_t(s) = 0$. In addition, at most one of $\triangle R_t^+(s-1,s)$ and $\triangle R_t^-(s-1,s)$ can be positive in a given period since their associated columns in the constraint matrix are linearly dependent. In the capacity constraint (2.4) $-\triangle R_t^+(s-1,s) + \triangle R_t^-(s-1,s) > -C + R_t^{'}(s-1)$ implies that the planned release changes are not constrained by capacity, and hence the associated dual variable $\sigma_t(s) = 0$. When $\triangle R_t^+(s-1,s) > 0$, we must have $\triangle R_t^+(s-1,s) > -R_t^{'}(s-1)$ implying $\eta_t(s) = 0$ from (2.5). For negative changes, we cannot reduce the planned release quantity by more than $R_t^{'}(s-1)$ units due to the nonnegativity of release changes. Hence, $\triangle R_t^-(s-1,s) < R_t^{'}(s-1)$ also implies $\eta_t(s) = 0$, unless demand is zero. $\triangle R_t^+(s-1,s) > 0$ also indicates $-\sigma_t(s) + \eta_(s) + \sum_{\tau=t+L}^{s+T-1} \beta_\tau(s) = Q_t^+(s)$ by constraint (2.11). $\triangle R_t^-(s-1,s) > 0$ implies $\sigma_t(s) - \eta_(s) - \sum_{\tau=t+L}^{s+T-1} \beta_\tau(s) = Q_t^-(s)$ by constraint (2.13). If the backlog variable $B_t(s) > 0$, then $\beta_t(s) = \varphi_t + \pi_t$ by constraint (2.10). These relationships will be used in the following sections to develop bounds on the values of the change costs that will guarantee freezing of schedules by eliminating planned release changes in a given period. Specifically, we seek the values of the unit change costs for each period that result in optimal values of zero for the planned release changes in that period.

In the next Sect. 2.3.5, we describe our implementation of freezing the schedule in the rolling horizon environment. In Sect. 2.4, we examine the behavior of release changes in a planning epoch followed by some insights into the effects of unit release change costs on the behavior of release changes from one epoch to the next. We then derive the change costs that can freeze the schedule in Sect. 2.5.

2.3.5 *Examples of Freezing the Schedules*

Freezing the schedule in a particular planning epoch s for a specific number of periods T such that $t = s, \ldots, s + T - 1$ means eliminating all planned release changes in these periods, causing all release change variables for those periods to take a value of zero. We seek the values of the unit change costs for each period that will result in optimal values of zero for the planned release changes in each period. We shall assume that if a given period $s+k$ is frozen, all periods $s, s+1, \ldots, s+k-1$ prior to it in the planning window must also be frozen.

Figure 2.2 shows an example of freezing the schedule of release plans for the third period $s + 2$ in epoch s in a $T = 7$ period planning window with a fixed lead time of $L = 1$ period. With a lead time of one period, we must release work in period $s + 2$ to meet demand in period $s + 3$. The orange colored cells labelled (L) indicate the period with fixed pre-release based on future demand outside the planning window. This period cannot be frozen because its releases must be planned

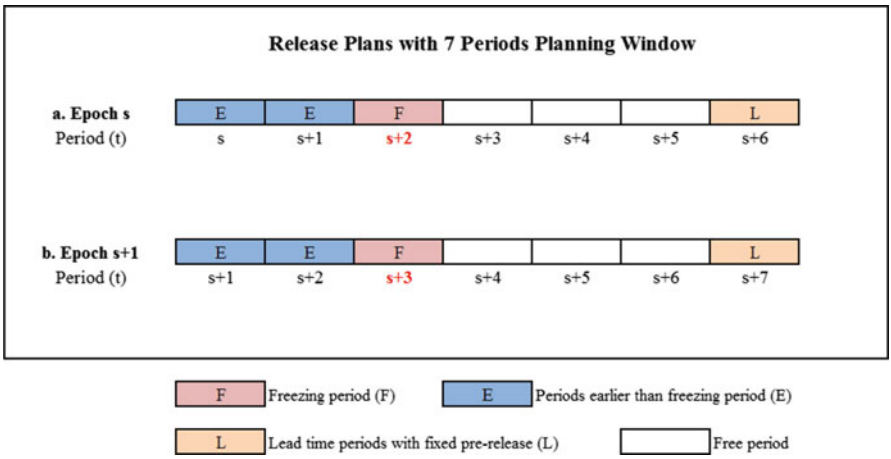


Fig. 2.2 Example of freezing the schedule

based on demand information outside the current planning window. The red cells labelled (F) indicate the periods we decide to freeze. For example, as Fig. 2.2a shows, when we freeze the release plan in period $s + 2$ in epoch s, we also need to freeze periods s and $s + 1$ prior to it which are labelled (E). In the next epoch $s + 1$ as shown in Fig. 2.2b, we will freeze period $s + 3$ and all periods prior to it in the current planning epoch.

2.4 Behavior of Positive and Negative Release Changes

In this section, we analyze the causes of release changes and how they affect the production system. We begin our analysis by assuming that the only source of variability from one planning epoch s to the next epoch $s + 1$ is the newly revealed demand information in the last period $s + T - 1$ of each epoch. This assumption implies that once the demand in a period is observed, it does not change. While this assumption is often not realistic—demand forecasts for a future period can usually be updated until the demand is realized—this simplified approach allows us to gain insight from the study of a simpler problem. Later in this section we relax this assumption, allowing demand changes at all periods within the current planning window.

It is intuitive that a sufficiently high unit release change cost should completely eliminate all release changes; a release cost of infinity corresponds to a constraint setting all planned release changes to zero. Setting the unit cost of positive release changes higher than the backlog cost will drive the primal model presented in Sect. 2.3 to hold backlogs instead of modifying releases. Negative release changes, on the other hand, may be eliminated by adjusting the associated change costs

relative to the unit FGI and WIP holding costs, causing the model to hold inventory instead of reducing releases. Thus, positive release change costs should take backlog cost into consideration, while negative release costs ought to be driven by the cost of holding FGI. Both positive and negative change costs should be related to the lead time and the timing of changes.

When producing with a lead time of L periods in a rolling horizon environment, we need to impose an ending condition on releases for future demand outside the current planning window in order to prevent the model from setting releases to zero and eliminating all production in periods beyond the current window. For there to be no release changes at all, the newly revealed demand $D_{s+T-1}(s)$ must be satisfied by the release quantities planned in the previous epoch. Thus, two general causes, excess demand and excess pre-release, lead to release changes as discussed in the subsequent sub-sections.

2.4.1 Excess Demand

If $D_{s+T-1} > R'_{s+T-1-L}(s-1)$ a positive release change is necessary for the new demand to be met without backlogging, increasing the total amount of material released in the planning window over that already planned for that period in the previous epoch $s-1$. In this case we have the following three scenarios based on constraint (2.4) with residual capacity:

- $D_{s+T-1}(s) - R'_{s+T-1-L}(s-1) \leq \sum_{t=s}^{s+T-1-L} \left[C - R'_t(s-1)\right]$

In this case the excess demand is less than the cumulative residual capacity, so the positive release change required to meet the excess demand is feasible and given by:

$$\sum_{t=s}^{s+T-1-L} \triangle R_t^+(s-1,s) = D_{s+T-1}(s) - R'_{s+T-1-L}(s-1). \tag{2.17}$$

Equation (2.17) implies, as expected, that excess demand in period $s+T-1$ may require positive planned release changes in period s to $s+T-1-L$ to satisfy the excess demand. This is because the excess capacity in period $s+T-1$ may not be sufficient for the necessary release changes, requiring us to release material earlier and hold it in FGI until period $s+T-1$.

- $D_{s+T-1}(s) - R'_{s+T-1-L}(s-1) > \sum_{t=s}^{s+T-1-L} \left[C - R'_t(s-1)\right] = 0$

In this case, there is no residual capacity available to produce any of the additional demand, all of which must be backlogged, yielding:

$$B_{s+T-1}(s) = D_{s+T-1}(s) - R'_{s+T-1-L}(s-1). \tag{2.18}$$

- $D_{s+T-1}(s) - R'_{s+T-1-L}(s-1) > \sum_{t=s}^{s+T-1-L} \left[C - R'_t(s-1)\right] > 0$

In this situation, the maximum amount of the incremental demand that can be accommodated with positive release changes is:

$$\sum_{t=s}^{s+T-1-L} \triangle R_t^+(s-1,s) = \sum_{t=s}^{s+T-1-L} \left[C - R'_t(s-1)\right], \tag{2.19}$$

and the remainder must be backlogged. However, the entire amount given by (2.19) need not necessarily be released; the model may decide to backlog some of this material if the cost of positive release changes is sufficiently high. In this case the backlogs will be at least:

$$B_{s+T-1}(s) \geq D_{s+T-1}(s) - R'_{s+T-1-L}(s) - \sum_{t=s}^{s+T-1-L} \left[C - R'_t(s-1)\right]. \tag{2.20}$$

2.4.2 Excess Pre-release

$D_{s+T-1}(s) < R'_{s+T-1-L}(s)$ implies that we planned to release more material in the previous planning epoch than the new demand requires; our initial estimate U of the demand in period $s+T-1$ has been revised down to a lower number. The optimal plan for epoch s will potentially have a negative release change in order not to carry unnecessary WIP and hold extra FGI. With an appropriately specified negative release change cost, the model will reduce releases only in period $s+T-1-L$ by the amount:

$$\triangle R_t^-(s-1,s) = \max\left[R'_{s+T-1-L} - D_{s+T-1}(s), 0\right] > 0. \tag{2.21}$$

In conclusion, the causes of positive and negative changes are different. Without positive release changes to satisfy excess demand, we will have backlogs, while without negative release changes to eliminate unnecessary material from the system, we will need to hold unnecessary inventory at additional cost. Thus, we should treat positive and negative changes in different ways, suggesting that they need to be assigned different unit change costs. When the only source of demand variability is the new demand information in the final period $s+T-1$ of the current epoch, we will only have negative changes in period $s+T-1-L$,since demand is not updated in periods earlier than period $s+T-1$. However, a high demand observation in the final period $s+T-1$ may cause positive release changes throughout the current planning window.

However, it is still not clear how to set change costs to guarantee freezing of the schedule in specific periods. In addition, planning in a rolling horizon environment complicates the cost settings: any decision in one epoch affects not just the current epoch, but potentially also those in later epochs. Hence we seek a lower bound on the values of change costs that guarantee freezing of the schedule in the current epoch. However, these results are valid only for decisions made in the current epoch. It remains possible that new demand information in some future epoch will call for positive or negative release changes, even with the costs we specify. We will begin with an analysis of negative release change costs, and then examine positive release change costs.

From this point onward, we relax the assumption that the only source of potential release changes is the new demand information in the end of each epoch; we will allow updated demand forecasts for each period in each epoch.

2.5 Release Change Costs for a Single Product

In this section, we present the analysis for positive and negative release changes from the last period $s + T - 1 - L$ of epoch s backward to its first period s.

2.5.1 Negative Release Change Costs

- **Freezing Period *s+T-1-L* When Excess Pre-release occurs in Epoch *s***

When there is excess pre-release in period $s + T - 1$, we will only have negative release changes in period $s + T - 1 - L$. A high enough negative release change cost that makes negative release changes unattractive results in $\triangle R^{-}_{s+T-1-L}(s-1, s) = 0$ so that constraint (2.13) takes the form:

$$\sigma_{s+T-1-L}(s) - \eta_{s+T-1-L}(s) - \beta_{s+T-1-L}(s) \leq \gamma^{-}_{s+T-1-L} - L\omega - \varphi. \tag{2.22}$$

Since we are considering a negative release change in a single-product model, the capacity constraint (2.4) in period $s + T - 1 - L$ is not binding, so $\sigma_{s+T-1-L}(s) = 0$. In order to derive a lower bound on the negative change cost that will result in zero negative changes at optimality, we note that the maximum value of $-\eta_{s+T-1-L}(s)$ is zero. $\triangle R^{-}_{s+T-1-L}(s-1, s) = 0$ in an optimal solution means carrying the excess pre-release as FGI, implying $I_{s+T-1-L}(s) > 0$. The complementary slackness condition applied to constraint (2.3) gives $\beta_{s+T-1-L}(s) = 0$. Thus the negative release change cost to guarantee freezing period $s + T - 1 - L$ in epoch s must satisfy:

$$\gamma^{-}_{s+T-1-L} \geq L\omega + \varphi. \tag{2.23}$$

Note, however, that freezing the schedule in period $s + T - 1 - L$ in epoch s will create an excess pre-release in period $s + T - 1$ in epoch $s + 1$, impacting decisions in the next epoch $s + 1$.

We now consider the negative change costs required to freeze the schedule in period $s + T - 2 - L$ in epoch s, given updated demand information in period $s + T - 2$ only.

- **Freezing Period *s+T-2-L* When Excess Pre-release occurs in Epoch *s***

In order to eliminate negative changes in period $s + T - 2 - L$ due to excess pre-release in period $s + T - 2$ in epoch s, the worst case situation is to carry all the excess pre-release as FGI for both periods $s + T - 2$ and $s + T - 1$. When $\triangle R^{-}_{s+T-2-L}(s-1, s) = 0$ at optimality the complementary slackness condition for (2.13) yields:

$$\begin{aligned} &\sigma_{s+T-2-L}(s) - \eta_{s+T-2-L}(s) - \sum_{\tau=s+T-2}^{s+T-1} \beta_\tau(s) \\ &\leq \gamma^{-}_{s+T-2-L} - L\omega - \sum_{\tau=s+T-2}^{s+T-1} \varphi. \end{aligned} \tag{2.24}$$

Since the capacity constraints are not binding in period $s + T - 2 - L$ and we wish to have zero negative release change in the optimal solution, we must have $\sigma_{s+T-2-L}(s) = 0$ by constraint (2.4). We set $-\eta_{s+T-2-L}(s) = 0$ to obtain a lower bound on the negative change cost. Freezing excess pre-releases results in positive FGI $I_{s+T-2}(s) > 0$, implying $\beta_{s+T-2}(s) = 0$ by (2.3) and thus:

$$\gamma^{-}_{s+T-2-L} \geq L\omega + 2\varphi - \beta_{s+T-1}(s). \tag{2.25}$$

Freezing excess pre-releases in period $s+T-2-L$ may affect the decision in period $s + T - 1 - L$ under the following two conditions:

1. $I_{s+T-1}(s) = 0$: $\beta_{s+T-1}(s) = L\omega + \varphi$
2. $I_{s+T-1}(s) > 0$: $\beta_{s+T-1}(s) = 0$

If eliminating negative release changes in period $s + T - 2 - L$ will not require holding FGI in period $s + T - 1 - L$ from (2.25) we obtain:

$$\gamma^{-}_{s+T-2-L} \geq \varphi. \tag{2.26}$$

On the other hand, if the freezing period $s+T-2-L$ requires holding FGI in period $s + T - 1$, we need higher negative change costs to freeze the schedule in period $s + T - 2 - L$, since the change cost must offset the additional cost of carrying the excess material for an additional period in period $s + T - 1 - L$, yielding:

$$\gamma^{-}_{s+T-2-L} \geq L\omega + 2\varphi. \tag{2.27}$$

Table 2.1 Negative release change costs required to freeze the schedule

Freezing the schedule in epoch s		
Period	Worse-case periods	Unit negative change cost
s	$T-L$	$\gamma_s^- \geq L\omega + (T-L)\varphi$
	$T-L-1$	$\gamma_s^- \geq (T-L-1)\varphi$
	$\vdots$	$\vdots$
	2	$\gamma_s^- \geq 2\varphi$
	1	$\gamma_s^- \geq \varphi$
$\vdots$	$\vdots$	$\vdots$
$s+T-2-L$	2	$\gamma_{s+T-2-L}^- \geq L\omega + 2\varphi$
	1	$\gamma_{s+T-2-L}^- \geq \varphi$
$s+T-1-L$	1	$\gamma_{s+T-1-L}^- \geq L\omega + \varphi$

Table 2.2 Negative release change costs guaranteeing freezing of plan

Freezing the plan in epoch s	
Period	Unit negative change cost
s	$\gamma_s^- \geq L\omega + (T-L)\varphi$
$\vdots$	$\vdots$
$s+T-2-L$	$\gamma_{s+T-2-L}^- \geq L\omega + 2\varphi$
$s+T-1-L$	$\gamma_{s+T-1-L}^- \geq L\omega + \varphi$

This is a lower bound on the negative release change cost required to guarantee freezing of the schedule under all circumstances. The approach can be applied to periods $s+T-3-L, \ldots, s$. We summarize the minimum unit negative release change costs required to freeze the schedule in each period of an epoch s in Table 2.1. In Table 2.1 "Worst-Case Periods" indicates the maximum number of periods for which FGI must be carried if negative release changes are eliminated in this period.

We emphasize once again that to guarantee elimination of negative changes in the current epoch under all circumstances, we need to set the change cost to at least the lower bound specified in Table 2.2. Recall also that these lower bounds only guarantee the freezing of the schedule in the current epoch, not that there will be no release changes in the specified period in any future epoch.

2.5.2 Positive Release Change Costs

When updated demand forecast information reveals increased demand, we need to use available residual capacity within the current planning window consisting of periods $s, \ldots, s+T-1-L$ to satisfy unmet demand and reduce backlogging. In a T period planning window with a lead time of L periods, it is quite possible to

have positive changes in all $T - L$ periods because of unmet excess demand. We will begin the analysis from period $s + T - 1$ with excess demand in epoch s first. We will then extend the analysis to the rest of the periods in epoch s.

- **Freezing Period *s+T-1-L* When Excess Demand Occurs in Period *s+T-1***

When there is excess demand in period $s + T - 1$, we may have positive release changes in all periods from period s to period $s + T - 1 - L$ depending on the availability of residual capacity in these periods. If there is no capacity available in period $s + T - 1 - L$, there will never be a positive release change for that period in an optimal solution. If there is residual capacity in that period, freezing period $s + T - 1 - L$ in epoch s implies $\triangle R^{+}_{s+T-1-L}(s-1, s) = 0$ and $B_{s+T-1}(s) > 0$, which in turn implies $\beta_{s+T-1}(s) = \pi + \varphi$ by (2.10). Applying the complementary slackness condition to constraint (2.12), we obtain:

$$-\sigma_{s+T-1-L}(s) + \eta_{s+T-1-L}(s) + \beta_{s+T-1}(s) \leq \gamma^{+}_{s+T-1-L} + L\omega + \varphi. \tag{2.28}$$

Since we freeze the positive changes for period $s + T - 1 - L$, the capacity constraint (2.4) is not binding, implying $\sigma_{s+T-1-L}(s) = 0$. Freezing the positive changes also makes constraint (2.5) not binding so that $\eta_{s+T-1-L}(s) = 0$ since $0 > -R^{'}_{s+T-1-L}(s)$. Based on these, we obtain a lower bound on the positive release change cost required to freeze positive changes in period $s + T - 1 - L$ as:

$$\gamma^{+}_{s+T-1-L} \geq \pi - L\omega. \tag{2.29}$$

- **Freezing Period *s+T-2-L* When Excess Demand Occurs in Period *s+T-2***

A positive release change in period $s + T - 2 - L$ may stem from excess demand in period $s + T - 1$, in period $s + T - 2$, or both. When we decide to eliminate positive release changes in period $s + T - 2 - L$, in the worst case we will cause two periods of backlogging in periods $s + T - 2$ and $s + T - 1$. Since freezing period $s + T - 2 - L$ in epoch s assumes there is residual capacity in that period, we have $\triangle R^{+}_{s+T-2-L}(s-1, s) = 0$ and $B_{s+T-2}(s) > 0$, which implies:

$$\begin{aligned} &-\sigma_{s+T-2-L}(s) + \eta_{s+T-2-L}(s) + \beta_{s+T-2}(s) + \beta_{s+T-1}(s) \\ &\quad \leq \gamma^{+}_{s+T-2-L} + L\omega + 2\varphi \end{aligned} \tag{2.30}$$

and $\beta_{s+T-2}(s) = \pi + \varphi$ by constraints (2.12) and (2.10). When freezing the schedule causes constraints (2.4) and (2.5) not to be binding, $\sigma_{s+T-2-L}(s) = \eta_{s+T-2-L}(s) = 0$. The positive release change cost to freeze period $s + T - 2 - L$ due to excess demand in period $s + T - 2$ in epoch s is obtained as:

$$\gamma^{+}_{s+T-2-L} \geq \pi - L\omega - \varphi + \beta_{s+T-1}(s). \tag{2.31}$$

Table 2.3 Positive release change costs to freeze the schedule

Freezing the schedule in epoch s		
Period	Worse-case periods	Unit positive change cost
s	$T-L$	$\gamma_s^+ \geq (T-L)\pi - L\omega$
	$T-L-1$	$\gamma_s^+ \geq (T-L-1)\pi$
	$\vdots$	$\vdots$
	2	$\gamma_s^+ \geq 2\pi$
	1	$\gamma_s^+ \geq \pi$
$\vdots$	$\vdots$	$\vdots$
$s+T-2-L$	2	$\gamma_{s+T-2-L}^+ \geq 2\pi - L\omega$
	1	$\gamma_{s+T-2-L}^+ \geq \pi$
$s+T-1-L$	1	$\gamma_{s+T-1-L}^+ \geq \pi - L\omega$

If freezing changes in period $s+T-2-L$ causes backlogging in period $s+T-1$, then $\beta_{s+T-1}(s) = \pi + \varphi$ by (2.10) so that:

$$\gamma_{s+T-2-L}^+ \geq 2\pi - L\omega. \tag{2.32}$$

If freezing changes in period $s+T-2-L$ does not cause backlogging and FGI in period $s+T-1$, then $\beta_{s+T-1}(s) = L\omega + \varphi$ yielding:

$$\gamma_{s+T-2-L}^+ \geq \pi. \tag{2.33}$$

If we want to guarantee freezing period $s+T-2-L$ in an epoch s, we need to set the positive change cost to at least $2\pi - L\omega$. The results of the same procedure applied to the rest of the analysis to eliminate positive release changes are summarized in Table 2.3. The "Worse-Case Periods" in Table 2.3 represents the maximum number of periods for which backlogs must be carried. For example, if we decide to freeze period s in epoch s, it may cause backlogging in up to $T-L$ periods. From Table 2.3, we observe that setting positive release change cost to eliminate positive release change in period t in epoch s must consider how freezing affects the future periods within the current planning epoch. The costs to guarantee elimination of positive changes for each period are summarized in Table 2.4. However, recall also that these lower bounds on the positive release change cost only guarantee the freezing of the schedule in the current epoch, and do not guarantee that there will be no release changes in a specified period in future epochs.

Table 2.4 Positive release change costs guaranteeing to freeze the schedule

Freezing the schedule in epoch s	
Period	Unit positive change cost
s	$\gamma_s^- \geq (T-L)\pi - L\omega$
$\vdots$	$\vdots$
$s+T-2-L$	$\gamma_{s+T-2-L}^- \geq 2\pi - L\omega$
$s+T-1-L$	$\gamma_{s+T-1-L}^- \geq \pi - L\omega$

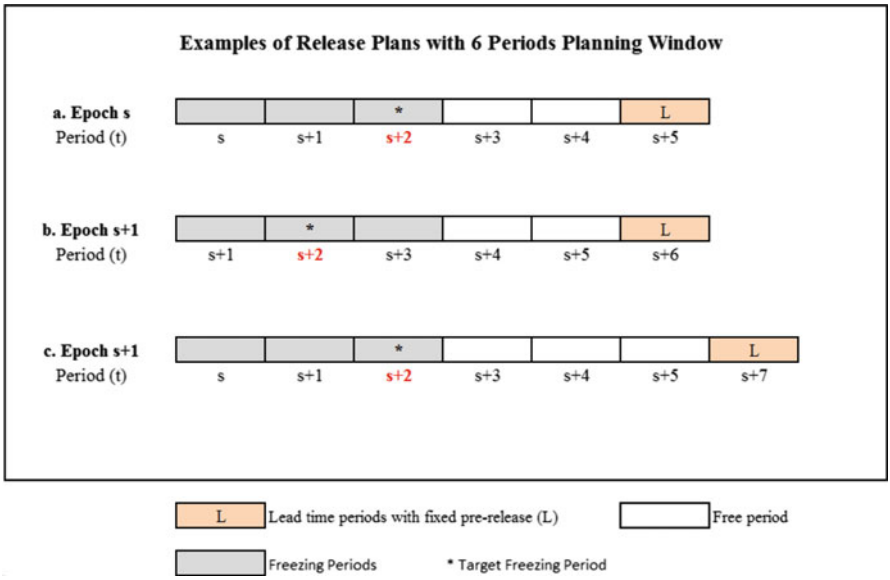


Fig. 2.3 An example of freezing the schedule across epochs

2.5.3 *Freezing Costs Inside an Epoch or Across Epochs*

In the previous two sub-sections, we have shown how to set positive and negative release change costs to guarantee freezing the schedule for a given period in a specific planning epoch s. For example, if we choose to freeze the schedule in period $s+2$ with $T=6$ planning periods in epoch s as Fig. 2.3a shown, we not only need to freeze period $s+2$ but also all periods preceding it in the epoch s. This means we need to freeze the three periods from period s to $s+2$ in each epoch s.

If we decide to freeze period $s+2$ in epoch s in an environment with lead time of $L=1$ period, we can use Tables 2.2 and 2.4 to set the change costs for period $s+2$ in epoch s as:

$$\gamma_{s+2}^+(s) \geq 3\pi - \omega \tag{2.34}$$

and

$$\gamma^{-}_{s+2}(s) \geq 3\varphi + \omega. \tag{2.35}$$

In the next epoch $s+1$ as shown in Fig. 2.3b, we also need to freeze the three periods $s+1$, $s+2$, and $s+3$. How can we set the change cost for period $s+2$ in epoch $s+1$? Based on Tables 2.2 and 2.4, we need to set the change cost for period $s+2$ in epoch $s+1$ as:

$$\gamma^{+}_{s+2}(s+1) \geq 4\pi - \omega \tag{2.36}$$

and

$$\gamma^{-}_{s+2}(s+1) \geq 4\varphi + \omega. \tag{2.37}$$

From (2.36) and (2.37), the associated positive and negative changes costs for period $s+2$ in epoch $s+1$ are higher than (2.34) and (2.35). This is because in a rolling horizon environment, we truncate the planning problem by considering information only within the current planning window of T periods. Thus, we will obtain lower change costs to guarantee freezing period $s+2$ in epoch s since we ignore the impact of release changes in the current epoch on periods outside the current planning window. If we extend the planning window length to $T = 7$ periods with $L = 1$ period in an epoch as shown in Fig. 2.3c, we still can use Tables 2.2 and 2.4 to set the change costs to freeze period $s+2$ in epoch s. Since we have one more period $s+6$ in epoch s, the associated change costs to freeze period $s+2$ are now:

$$\gamma^{+}_{s+2}(s) \geq 4\pi - \omega \tag{2.38}$$

and

$$\gamma^{-}_{s+2}(s) \geq 4\varphi + \omega. \tag{2.39}$$

The change costs to freeze period $s+2$ in this condition are equal to those in a six period example in epoch $s+1$. In summary, the freezing decision affects not only planning periods in the current epoch but also future periods as yet outside the current planning window. However, we cannot assess the impact of freezing decisions on periods currently outside the planning window until we know their demand information.

2.6 Numerical Examples

In this section, we present numerical examples for the single-product model with fixed lead time and capacity analyzed in the previous sections.

Table 2.5 Costs of single-product model

Backlog cost (π)	FGI cost (φ)	WIP cost (ω)
\$110	\$12	\$6

2.6.1 Settings and Assumptions

We assume that the length of the planning window in a planning epoch is $T = 6$ periods. The overall planning period is 400 periods, giving 395 planning epochs. The overall capacity for each period is 70 units. For each planning epoch, we allow updated demand forecast for all periods. We also assume the demand is normally distributed with mean of 60 units and standard deviation of 27 units, implying an average capacity utilization of 0.86. We generate the demand by using Arena Input Analyzer without allowing negative values. The lead time is $L = 2$ periods and producing one unit of output requires one unit of capacity. The associated cost values are shown in Table 2.5.

In this section we want to confirm that the derived lower bounds on the release change costs can guarantee freezing of the schedule under all circumstances. In our example in any epoch s, periods $s, \ldots, s + 3$ may have release changes so that we can define the overall positive changes in any period s as:

$$\triangle R^{+}(s) = \sum_{s} \triangle R_{s}^{+}(s-1, s), \forall s \in (1, K - T + 1). \tag{2.40}$$

If we want to measure all positive release changes in any period $s + 3$ in any epoch s, the latest period in which release changes can take place, we can define as:

$$\triangle R^{+}(s+3) = \sum_{s} \triangle R_{s+3}^{+}(s-1, s), \forall s \in (1, K - T + 1). \tag{2.41}$$

Similar expressions can be defined to compute the analogous quantities for negative release changes.

2.6.2 Examples of Setting Release Change Costs

Since we have a common lead time of $L = 2$ periods, we may have release changes in periods s through $s + 3$ in epoch s. Thus, from Tables 2.4 and 2.2 we can set the release change costs as shown in Table 2.6. From Table 2.6, positive change cost is significantly higher than negative change cost in each period and freezing earlier periods in an epoch requires higher change costs. For example, if we want to freeze period $s + 1$ in epoch s, we must set positive and negative change costs to \$318 and \$48, respectively. In the next epoch $s + 1$, can we still set the release change costs for period $s + 1$ in this manner to guarantee freezing? The answer is "No."

Table 2.6 Release change cost in epoch s

	Epoch/Period	s	s+1	s+2	s+3	s+4	s+5
Positive	s	\$428	\$318	\$208	\$98		
Negative	s	\$60	\$48	\$36	\$24		

Table 2.7 Release change cost in epoch $s + 1$

	Epoch/Period	s	s+1	s+2	s+3	s+4	s+5
Positive	s	\$428	\$318	\$208	\$98		
	s+1		\$428	\$318	\$208	\$98	
Negative	s	\$60	\$48	\$36	\$24		
	s+1		\$60	\$48	\$36	\$24	

Table 2.8 Release change cost when $T = 7$ periods in epoch s

	Length/Period	s	s+1	s+2	s+3	s+4	s+5
Positive	$T = 6$	\$428	\$318	\$208	\$98		
	$T = 7$	\$538	\$428	\$318	\$208	\$98	
Negative	$T = 6$	\$60	\$48	\$36	\$24		
	$T = 7$	\$72	\$60	\$48	\$36	\$24	

From the analysis in Sect. 2.5.3, we must increase the associated release change costs to guarantee freezing of the schedule in epoch $s + 1$. Table 2.7 clearly shows how to set the release change costs from epoch s to epoch $s + 1$. For example, we need to increase both change costs in period $s + 1$ to \$428 and \$60, respectively, in contrast to \$318 and \$48 for this period under the previous planning epoch s. This is because we truncate the infinite horizon planning problem into a sequence of finite horizon problems. In epoch s, period $s + 1$ does not need to consider the effect of freezing on period $s + 4$; however, in epoch $s + 1$, we need to consider the period $s + 4$ that causes the increase in costs. We also see a similar trend in release change costs on other periods in different planning epochs.

When we extend the planning window length from $T = 6$ to $T = 7$ periods, in epoch s, we also need to increase the associated release change costs for periods s to $s + 3$. Table 2.8 shows the cost settings for planning window lengths of $T = 6$ to $T = 7$ periods based on the analysis in Sect. 2.5.3. When we have one more period in an planning epoch, we need to increase the positive and negative release change costs, for example in period s, from \$428 and \$60 to \$538 and \$72, respectively.

By setting the release change costs slightly higher than the derived release change costs to prevent multiple optimal solutions, we can guarantee elimination of either positive or negative release changes. Figure 2.4 presents the results of release changes when we apply different change costs to freeze different periods in any epoch. Blue bars represent the results of positive release changes and red bars represent the results of negative release changes.

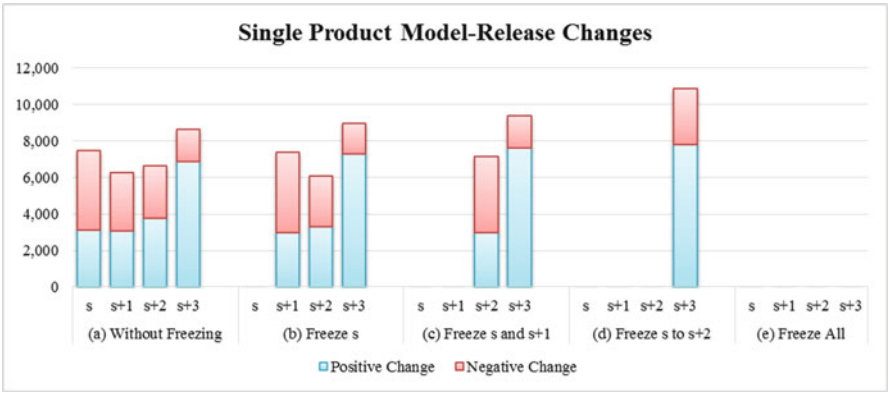


Fig. 2.4 Results of freezing single product

Figure 2.4a shows that when there are no change costs we observe release changes in all periods. When we set the change costs using the derived lower bounds, we can eliminate all release changes in period s, which is shown in Fig. 2.4b. When we want to freeze periods s and $s+1$, the derived lower bound for period $s+1$ also holds. We find zero release change in period $s+1$ as shown in Fig. 2.4c. We also find the same results for guaranteeing elimination of release changes when we set the change costs equal to the derived lower bounds to freeze period $s+2$ and $s+3$ in Fig. 2.4d, e. Thus, the numerical results confirm that we can freeze the schedule by setting the release change costs to the derived lower bound.

2.7 Conclusion

In this chapter we have analyzed the relation between two different approaches for improving schedule stability, the use of change costs to penalize planned changes and the freezing of the plan in certain periods by prohibiting any planned changes. We formulate the planning problem to be solved at each epoch as a linear program, and analyze the structure of the optimal solutions to derive lower bounds on the values of the unit change costs that will ensure zero release changes. We find that the unit change costs required to ensure freezing in a given period is lower for later periods in the epoch. This is intuitive since any excess inventory and backlogs associated with earlier periods in the epoch will be held longer. We also find that freezing positive release changes require higher unit change costs than freezing negative changes, since the former are driven by backlog costs and the latter by inventory holding costs.

Although the production system we have considered is very simple compared to practical industrial systems, we believe this work provides useful insights. First of all, it allows a rough-cut analysis of the change costs required to ensure schedule

freezing, allowing management to assess at least qualitatively, whether they believe the impact of the planned changes will indeed result in costs of this magnitude. The formulation of the planning problem at each epoch in terms of planned changes rather than gross release quantities also provides a basis for further analysis of the problem. Of particular interest is the extension of the analysis in this chapter to systems with multiple products. In this case we conjecture that the unit costs derived in this paper will not be sufficient to eliminate all planned changes, since the change costs must also offset the benefit obtained by reallocating capacity between products as demand information is updated. The analysis applied in this paper is based on the optimality conditions for linear programs, which cannot be applied to problems with setup times that require mixed-integer programming formulations. Nevertheless, the ideas from this work could be applied numerically to obtain estimates of change costs that would freeze schedules in this environment also. Finally, the extension of this approach to multiple stage production systems involving different capacitated resources is also of interest.

Acknowledgements This research was supported by the National Science Foundation under Grant No.1029706. The opinions expressed in the article are those of the authors and do not represent the views of the National Science Foundation.

References

Aouam T, Uzsoy R (2014) Zero-order production planning models with stochastic demand and workload-dependent lead times. Int J Prod Res 1–19. ISSN 0020-7543

Bai X, Davis JS, Kanet JJ, Cantrell S, Patterson JW (2002) Schedule instability, service level and cost in a material requirements planning system. Int J Prod Res 40(7):1725–1758. ISSN 0020-7543

Blackburn JD, Kropp DH, Millen RA (1985) MRP system nervousness: causes and cures. Eng Costs Prod Econ 9(1–3):141–146. ISSN 0167188X

Blackburn JD, Kropp DH, Millen RA (1986) A comparison of strategies to dampen nervousness in MRP systems. Manag Sci 32(4):413–429. ISSN 0025-1909

Bookbinder JH, Tan JY (1988) Strategies for the probabilistic lot-sizing problem with service-level constraints. Manag Sci 34(9):1096–1108

Braun MW, Schwartz JD (2012) A control theoretic evaluation of schedule nervousness suppression techniques for master production scheduling. In: Decision policies for production networks. Springer, London, pp 143–171

Carlson RC, Jucker JV, Kropp DH (1979) Less nervous MRP systems: a dynamic economic lot-sizing approach. Manag Sci 25(8):754–761. ISSN 0025-1909

Carlson RC, Beckman SL, Kropp DH (1982) The effectiveness of extending the horizon in rolling production scheduling. Decis Sci 13(1):129–146. ISSN 0011-7315

Eilon S (1975) Five approaches to aggregate production planning. IIE Trans 7(2):118–131. ISSN 0569-5554

Fisher M, Ramdas K, Zheng Y (2001) Ending inventory valuation in multiperiod production scheduling. Manag Sci 47(5):679–692. ISSN 0025-1909

Grinold RC (1980) Time horizons in energy planning models. In: Energy policy modeling: United States and Canadian experiences. Springer, Netherlands, pp 216–232

Grinold RC (1983) Model building techniques for the correction of end effects in multistage convex programs. Oper Res 31(3):407–431

Hung Y, Leachman RC (1996) A production planning methodology for semiconductor manufacturing based on iterative simulation and linear programming calculations. IEEE Trans Semicond Manuf 9(2):257–269

Johnson LA, Montgomery DC (1974) Operations research in production planning, scheduling, and inventory control. Wiley, New York

Kadipasaoglu SN, Sridharan V (1995) Alternative approaches for reducing schedule instability in multistage manufacturing under demand uncertainty. J Oper Manag 13(3):193–221

Kropp DH, Carlson RC (1984) A lot-sizing algorithm for reducing nervousness in MRP systems. Manag Sci 30(2):240–244. ISSN 0025-1909

Kropp DH, Carlson RC, Jucker JV (1983) Heuristic lot-sizing approached for dealing with MRP system nervousness. Decis Sci 14(2):152–169

Lin N, Krajewski LJ, Leong GK, Benton WC (1994) The effects of environmental factors on the design of master production scheduling systems. J Oper Manag 11(4):367–384

Mather H (1977) Reschedule the reschedules you just rescheduled: way of life for MRP? Prod Invent Manag 18(1):60–79

Metters R, Vargas V (1999) A comparison of production scheduling policies on costs, service level, and schedule changes. Prod Oper Manag 8(1):76–91. ISSN 10591478

Missbauer H, Uzsoy R (2011) Optimization models of production planning problems. In: Kempf KG, Keskinocak P, Uzsoy R (eds) Planning production and inventories in the extended enterprise: a state of the art handbook. Springer, New York, pp 437–508

Narayanan A, Robinson P (2010) Evaluation of joint replenishment lot-sizing procedures in rolling horizon planning systems. Int J Prod Econ 127(1):85–94

Ravindran A, Kempf KG, Uzsoy R (2011) Production planning with load-dependent lead times and safety stocks for a single product. Int J Plan Sched 1(1/2):58. ISSN 2044-494X

Sahin F, Narayanan A, Robinson EP (2013) Rolling horizon planning in supply chains: review, implications and directions for future research. Int J Prod Res 51(18):5413–5430

Sridharan V, Berry WL (1990) Freezing the master production schedule under demand uncertainty. Decis Sci 21(1):97–120

Sridharan V, LaForge RL (1989) The impact of safety stock on schedule instability, cost and service. J Oper Manag 8(4):327–347. ISSN 02726963

Sridharan V, LaForge RL (1994a) Freezing the master production schedule: implications for fill rate. Decis Sci 25(3):461–469

Sridharan V, LaForge RL (1994b) A model to estimate service levels when a portion of the master production schedule is frozen. Comput Oper Res 21(5):477–486. ISSN 03050548

Sridharan V, Berry WL, Udayabhanu V (1987) Freezing the master production schedule under rolling planning horizons. Manag Sci 33(9):1137–1149. ISSN 0025-1909

Sridharan SV, Berry WL, Udayabhanu V (1988) Measuring master production schedule stability under rolling planning horizons. Decis Sci 19(1):147–166

Tarim SA, Kingsman BG (2004) The stochastic dynamic production/inventory lot-sizing problem with service-level constraints. Int J Prod Econ 88(1):105–119

Voss S, Woodruff DL (2006) Introduction to computational optimization models for production planning in a supply chain. Springer, Berlin

Wagner HM, Whitin TM (1958) Dynamic version of the economic lot size model. Manag Sci 5(1):89–96. ISSN 0025-1909

Xie J, Zhao X, Lee TS (2003) Freezing the master production schedule under single resource constraint and demand uncertainty. Int J Prod Econ 83(1):65–84

Yano CA, Carlson RC (1987) Interaction between frequency of rescheduling and the role of safety stock in material requirements planning systems. Int J Prod Res 25(2):221–232

Zhao X, Lam K (1997) Lot-sizing rules and freezing the master production schedule in material requirements planning systems. Int J Prod Econ 53(3):281–305

Zhao X, Lee TS (1993) Freezing the master production schedule for material requirements planning systems under demand uncertainty. J Oper Manag 11(2):185–205

Zhao X, Xie J (1998) Multilevel lot-sizing heuristics and freezing the master production schedule in material requirements planning systems. Prod Plan Control 9(4):371–384

Chapter 3
Stochastic Scheduling for a Network of Flexible Job Shops

Subhash C. Sarin, Hanif D. Sherali, Amrusha Varadarajan, and Lingrui Liao

Abstract In this chapter, we address the problem of optimally routing and sequencing a set of jobs over a network of flexible machines for the objective of minimizing the sum of completion times and the cost incurred, assuming stochastic job processing times. This problem is of particular interest for the production control in high investment, low volume manufacturing environments, such as pilot-fabrication of microelectromechanical systems (MEMS) devices. We model this problem as a two-stage stochastic program with recourse, where the first-stage decision variables are binary and the second-stage variables are continuous. This basic formulation lacks relatively complete recourse due to infeasibilities that are caused by the presence of re-entrant flows in the processing routes, and also because of potential deadlocks that result from the first-stage routing and sequencing decisions. We use the expected processing times of operations to enhance the formulation of the first-stage problem, resulting in good linear programming bounds and inducing feasibility for the second-stage problem. In addition, we develop valid inequalities for the first-stage problem to further tighten its formulation. Experimental results are presented to demonstrate the effectiveness of using these strategies within a decomposition algorithm (the L-shaped method) to solve the underlying stochastic program. In addition, we present heuristic methods to handle large-sized instances of this problem and provide related computational results.

Keywords Stochastic scheduling • Flexible job shop • Multi-site scheduling • L-shaped method • Branch-and-bound

S.C. Sarin (✉) • H.D. Sherali • A. Varadarajan • L. Liao
Grado Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, VA 24061, USA
e-mail: sarins@vt.edu

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_3

3.1 Introduction: Problem Statement and Related Literature

The production in high investment, low volume manufacturing environments, such as pilot-fabrication of microelectromechanical systems (MEMS) devices, gives rise to several special features of the underlying scheduling problem. Due to high prices of the processing equipments and complicated fabrication processes, it is impractical to assign dedicated equipment to each processing step. The relatively low volume of production during the pilot stage also implies that machine flexibility is highly desirable so that multiple product types can share processing equipments. Hence, each manufacturing facility is organized as a flexible job shop, serving multiple processing routes with flexible machines. Furthermore, due to novelty of the products and fabrication processes, a single facility often lacks the capability of performing all the required processing steps for a product from start to finish. To satisfy these special requirements, multiple manufacturing facilities are usually organized into a distributed fabrication network, where a central service provider coordinates production activities across facilities and directly deals with customers' requirements. Products are shipped from one facility to another until all processing requirements are met. For a given processing step, there may be multiple facilities that can provide the required service. The flexibility of cross-facility routing not only provides more pricing and quality options for the customers, but also makes transportation time and cost an important aspect of the scheduling problem. We designate this type of distributed fabrication network as the *Network of Flexible Job Shops* (**NFJS**).

The management of operations for an NFJS involves two types of decisions: (1) choosing a facility for each job operation (i.e., processing step) and assigning it to a compatible machine within the facility (i.e., *routing*) and (2) stipulating a processing sequence for the operations assigned to any given machine (i.e., *sequencing*). The routing decisions need to take transportation time and cost into consideration, as they can be quite significant between geographically dispersed facilities. On the other hand, sequencing decisions need to account for the fact that sequence-dependent set-up times are required to prepare for the processing of operations of different jobs on the same machine. In view of the pricing and quality options that are available in an NFJS, the customer specifies for each job a fixed budget, which can only be exceeded under a given penalty rate. The job arrival times, number of operations for each job, machines capable of processing each operation, transportation times and transportation costs between facilities, sequence-dependent set-up times, and customer budgets are assumed to be known (and thus, deterministic). On the other hand, since exact values of processing times are expected to vary due to the novelty of fabrication technologies, they are assumed to be stochastic. The problem that we address in this chapter can be succinctly stated as follows:

> Given a set of jobs and a network of flexible job shops, where operation processing times are uncertain but the sequence in which to process the operations of each job is known a priori, determine an allocation of job operations to facilities and a sequence in which to

process these operations on the machines in the facility so as to minimize a function of the completion times and the transportation and processing costs incurred.

The NFJS problem combines the characteristics of three well-known problems: the multi-site planning and scheduling problem, the flexible job shop scheduling problem, and the stochastic job shop scheduling problem. Multi-site *planning* problems are extensions of capacitated lot-sizing problems, with emphasis on transportation requirements and site-specific holding cost. Production is assigned to machines at multiple sites to satisfy demands during each period of the time horizon. The multi-site *scheduling* problem further addresses underlying production issues, such as inventory interdependency and change-over setup. To deal with the integrated multi-site planning and scheduling problem, iterative methods have been applied that alternate between solving the long-term planning problem and solving the short-term scheduling problem (see, for example, Roux et al. 1999; Guinet 2001; Gnoni et al. 2003). Others have considered the monolithic approach, either using the approach of variable time scale (Timpe and Kallrath 2000; Lin and Chen 2006), or relying on heuristic methods (Gascon et al. 1998; Sauer et al. 2000; Jia et al. 2003) to handle the resulting complexity. Lee and Chen (2001) provided a comprehensive study on scheduling with transportation considerations for the single facility environment. They considered two particular cases pertaining to transportation within a flow shop environment and transportation during final product distribution. To the best of our knowledge, no previous research in the multi-site planning and scheduling area has considered routing flexibility and stochastic processing times, both of which are very pertinent to the NFJS problem.

In a flexible job shop environment, for each processing step of a job, there are multiple alternative machines that are capable of providing the required service. Various methods have been applied to solve problems of this type. For example, Iwata et al. (1980) and Kim (1990) have considered dispatching rules; Nasr and Elsayed (1990) have applied greedy heuristic methods, Hutchison et al. (1991) have devised a hierarchical decomposition method that determines the assignment of operations to machines and then generates sequences. With regard to iterative local search methods, Brandimarte (1993) considered re-assignment and re-sequencing as two different types of moves, while Dauzère-Pérès and Paulli (1997) and Mastrolilli and Gambardella (2000) did not explicitly treat them as different. Subramaniam et al. (2000) have performed a simulation study with dynamic job arrival and machine break downs. One can also find applications of meta-heuristic methods to solve this problem including, but not limited to, particle swam optimization (Xia and Wu 2005) and genetic algorithms (Pezzella et al. 2008 and Wang et al 2005). The routing flexibility that characterizes the flexible job shop problem is also present in the NFJS problem, but with an important distinction that the alternative machines may be located at different facilities (sites), thereby requiring consideration of transportation time and cost into the scheduling problem.

The stochastic scheduling problem has been addressed in the literature in the classical flow shop and job shop environments. Optimal policies, dominance relations, and dispatching rules for two- and three-machine flow shop scheduling

problems having stochastic processing times have been developed by Ku and Niu (1986), Weiss (1982), Mittal and Bagga (1977), Cunningham and Dutta (1973), Bagga (1970), Talwar (1967), Makino (1965), Prasad (1981), Forst (1983), Pinedo (1982), Jia (1998), Elmaghraby and Thoney (1999), and Kamburowski (1999, 2000). These studies vary either in the distribution of the processing times used, or in the objective function, or in the amount of intermediate storage available between machines. Optimal rules have also been developed by Foley and Suresh (1984) and Pinedo (1982) to minimize the expected makespan for the m-machine flow shop problem with stochasticity in processing times. For work in stochastic job shops, see Golenko-Ginzburg et al. (1995, 1997, 2002), Singer (2000), Luh et al. (1999), Kutanoglu and Sabuncuoglu (2001), Yoshitomi (2002), Lai et al. (2004), and Tavakkoli-Moghaddam et al. (2005).

The remainder of this chapter is organized as follows. In Sect. 3.2, we model the NFJS problem as a two-stage stochastic program and present the L-shaped method for its solution. Besides developing the pertinent feasibility and optimality cuts, we also introduce an alternative approach to induce second-stage feasibility. In Sect. 3.3, the formulation of the first-stage problem is further tightened by using three types of valid inequalities, all of which rely upon the special structure of the NFJS problem. Computational results are provided in Sect. 3.4 to demonstrate the efficacy of our model formulation and solution approach. For even large-sized problem instances, we present heuristic methods and the results on their performances in Sect. 3.5. Concluding remarks are made in Sect. 3.6.

3.2 Stochastic Model for a Network of Flexible Job Shops

We model the stochastic NFJS problem as a two-stage stochastic program with recourse, where the first-stage variables are binary and pertain to the assignment of job operations to machines and to the sequencing of job operations for processing on these machines, while the second-stage variables are continuous and relate to the completion times and budget over-runs of the jobs, and where the uncertainty in processing time durations influences the job completion times. Multiple facilities are incorporated in our formulation by assigning to each machine a unique identification number that distinguishes it from the other machines in all the facilities, and by appropriately considering the inter-machine transportation times and costs. Stochastic processing times are modeled by a finite set of scenarios for the entire problem, and each of these scenarios assigns durations to every possible processing step and has an associated probability value.

We present the overall problem formulation and decompose it into two stages using Benders' decomposition (Benders 1962). Besides constructing the feasibility cuts and optimality cuts, we further reinforce the first-stage problem by including additional valid inequalities that induce feasibility in the second stage.

3.2.1 Model Formulation for the NFJS Problem

Notation

Indices:

Job index: $i = 1, \ldots, N$
Operation index for job i: $j = 1, \ldots, J_i$
Machine index: $m = 1, \ldots, |M|$ (where M is the set of machines)
Scenario index: $s = 1, \ldots, S$

Decision Variables

$$x^m_{(i,j)} = \begin{cases} 1, & \text{if operation } j \text{ of job } i \text{ is assigned to machine } m, \\ 0, & \text{otherwise.} \end{cases}$$

$$y^m_{(i,j,k,l)} = \begin{cases} 1, & \text{if operation } j \text{ of job } i \text{ directly precedes operation } l \text{ of} \\ & \text{job } k \text{ on machine } m, \\ 0, & \text{otherwise.} \end{cases}$$

$$v^{(e,f)}_{(i,j,j+1)} = \begin{cases} 1, & \text{if operation } j \text{ of job } i \text{ is performed on machine } e \text{ and} \\ & \text{operation } j+1 \text{ of job } i \text{ is performed on machine } f, \\ 0, & \text{otherwise.} \end{cases}$$

$t^s_{(i,j)}$ = completion time of operation j of job i under scenario s.
Δ^s_i = budget over-run for job i under scenario s.

Parameters

$H^{(m,s)}_{(i,j,k,l)}$ = an appropriately large positive number; its value is specified in (3.16) below.
w_i = number of parts in job i.
M = set of all the machines.
ϕ_s = probability of occurrence for scenario s.
$c^m_{(i,j)}$ = cost per unit processing time of operation j of job i on machine m.
$p^{(m,s)}_{(i,j)}$ = processing time of operation j of job i on machine m under scenario s.
Z_m = set of job operations that can be processed on machine m.
$M_{(i,j)}$ = set of machines capable of processing operation j of job i.
$u^m_{(i,j,k,l)}$ = changeover time to switch from operation j of job i to operation l of job k on machine m.
b_i = budget for job i.
r_i = ready time for the first operation of job i.
$d_{(e,f)}$ = transportation time between machines e and f.
$q_{(e,f)}$ = per part transportation cost between machines e and f.
α_i = cost coefficient for job i that is ascribed to its completion time.
β_i = penalty coefficient for job i corresponding to its budget over-run.

Formulation NFJSP

$$\begin{aligned}\text{Minimize } z = &\sum_{i=1}^{N}\alpha_i\left(\sum_{s=1}^{S}\phi_s t^s_{(i,J_i)}\right)+\sum_{i=1}^{N}\beta_i\left(\sum_{s=1}^{S}\phi_s\Delta_i^s\right)\\ &+\sum_{s=1}^{S}\phi_s\sum_{i=1}^{N}\sum_{j=1}^{J_i}\sum_{m\in M_{(i,j)}}p^{(m,s)}_{(i,j)}x^m_{(i,j)}\\ &+\sum_{m\in M}\sum_{(i,j)\in Z_m}\sum_{\substack{(k,l)\in Z_m\\(k,l)\neq(i,j)}}u^m_{(i,j,k,l)}y^m_{(i,j,k,l)}\\ &+\sum_{i=1}^{N}\sum_{j=1}^{J_i-1}\sum_{e\in M_{(i,j)}}\sum_{f\in M_{(i,j+1)}}d_{(e,f)}v^{(e,f)}_{(i,j,j+1)}\end{aligned}\tag{3.1}$$

subject to:

$$\begin{aligned}&t^s_{(i,j)}+\sum_{m\in M_{(i,j+1)}}\left(p^{(m,s)}_{(i,j+1)}x^m_{(i,j+1)}\right)+\sum_{e\in M_{(i,j)}}\sum_{f\in M_{(i,j+1)}}d_{(e,f)}v^{(e,f)}_{(i,j,j+1)}\\ &\quad\le t^s_{(i,j+1)},\quad \forall i=1,\ldots,N,\ j=1,\ldots,J_i-1,\ s=1,\ldots,S\end{aligned}\tag{3.2}$$

$$r_i+\sum_{m\in M_{(i,1)}}\left(p^{(m,s)}_{(i,1)}x^m_{(i,1)}\right)\le t^s_{(i,1)},\quad \forall i=1,\ldots,N,\ s=1,\ldots,S\tag{3.3}$$

$$\begin{aligned}t^s_{(i,j)}+p^{(m,s)}_{(k,l)}+u^m_{(i,j,k,l)}\le t^s_{(k,l)}+\left(1-y^m_{(i,j,k,l)}\right)H^{(m,s)}_{(i,j,k,l)},\\ \forall m\in M,\forall\,(i,j)\neq(k,l)\in Z_m,\ \forall s=1,\ldots,S\end{aligned}\tag{3.4}$$

$$\sum_{m\in M_{(i,j)}}x^m_{(i,j)}=1,\quad \forall i=1,\ldots,N,\ j=1,\ldots,J_i\forall i=1,\ldots,N,\ j=1,\ldots,J_i\tag{3.5}$$

$$\sum_{\substack{(i,j)\in Z_m\\(i,j)\neq(k,l)}}y^m_{(i,j,k,l)}\le x^m_{(k,l)},\quad \forall k=1,\ldots,N,\ l=1,\ldots,J_k,\ m\in M_{(k,l)}\tag{3.6}$$

$$\sum_{\substack{(i,j)\in Z_m\\(i,j)\neq(k,l)}}y^m_{(k,l,i,j)}\le x^m_{(k,l)},\quad \forall k=1,\ldots,N,\ l=1,\ldots,J_k,\ m\in M_{(k,l)}\tag{3.7}$$

$$\sum_{(i,j)\in Z_m} \sum_{\substack{(k,l)\in Z_m \\ (k,l)\neq(i,j)}} y^m_{(i,j,k,l)} \geq \sum_{(i,j)\in Z_m} x^m_{(i,j)} - 1, \quad \forall\ m \in M \tag{3.8}$$

$$\sum_{j=1}^{J_i} \sum_{m\in M_{(i,j)}} c^m_{(i,j)} p^{(m,s)}_{(i,j)} x^m_{(i,j)} + \left(\sum_{j=1}^{J_i-1} \sum_{e\in M_{(i,j)}} \sum_{f\in M_{(i,j+1)}} q_{(e,f)} v^{(e,f)}_{(i,j,j+1)} \right) w_i - \Delta^s_i \leq b_i,$$
$$\forall i = 1,\ldots,N,\ \ s = 1,\ldots,S \tag{3.9}$$

$$v^{(e,f)}_{(i,j,j+1)} \leq x^e_{(i,j)},\ v^{(e,f)}_{(i,j,j+1)} \geq x^e_{(i,j)} + x^f_{(i,j+1)} - 1,$$
$$\forall i = 1,\ldots,N, j = 1,\ldots,J_i - 1, e \in M_{(i,j)}, f \in M_{(i,j+1)} \tag{3.10}$$

$$\Delta^s_i \geq 0, \quad \forall i = 1,\ldots,N, s = 1,\ldots,S \tag{3.11}$$

$$x^m_{(i,j)} \in \{0,1\}, \quad \forall i = 1,\ldots,N, j = 1,\ldots,J_i, m \in M_{(i,j)} \tag{3.12}$$

$$y^m_{(i,j,k,l)} \in \{0,1\}, \quad \forall m = 1,\ldots,M, \forall\ (i,j) \neq (k,l) \in Z_m \tag{3.13}$$

$$v^{(e,f)}_{(i,j,j+1)} \in [0,1], \quad \forall i = 1,\ldots,N, j = 1,\ldots J_i - 1, e \in M_{(i,j)}, f \in M_{(i,j+1)}. \tag{3.14}$$

The objective function (3.1) is composed of five terms. The first and the second terms penalize the sum of job completion times and budget over-runs, respectively. The penalty coefficients reflect the customer's emphasis on the lead-time and costs incurred, and they also scale the first two terms to be commensurate with the next three terms, which are time based. The third term represents expected processing time for the operations of all the jobs; the fourth term computes the total set-up time on the machines, and the final term determines the sum of travel times incurred by all the jobs. Note that the last three terms in the objective function support the first term by aiding the achievement of lower completion times, while at the same time, reflect costs incurred by consuming machine and transportation capacities of the system. Constraints (3.2) capture precedence relationships between operations of the same job. Specifically, they state that under each scenario *s,* the completion time of operation $j+1$ of job i, $\forall i = 1,\ldots,N$, must be at least equal to the completion time of operation j of that job plus the processing time of operation $j+1$ and any travel time incurred between the two operations (set-up time is assumed to be job-detached, and hence, is not included here). Constraints (3.3) ensure (for each scenario) that each job does not commence its first operation earlier than its ready time. Constraints (3.4) establish relationships among the operations to be performed

on the same machine. Given two distinct job-operations, say (i, j) and (k, l) in Z_m for a certain machine m, if (i, j) were to directly precede (k, l), (i.e., $y^m_{(i,j,k,l)} = 1$), then the completion time of (k, l) under any scenario s must be at least equal to the completion time of (i, j) for that scenario, plus the processing time of (k, l) and the sequence-dependent set-up time between the two operations. Observe that when $y^m_{(i,j,k,l)} = 0$, i.e., (i, j) does not directly precede (k, l), the constraint becomes redundant by the choice of a suitably large value of $H^{(m,s)}_{(i,j,k,l)}$ (see (3.16) below). Constraints (3.5) ensure that each job-operation is assigned to exactly one machine out of the several alternative machines that can process it. Constraints (3.6) and (3.7) state that if a job-operation, say (k, l), is assigned to a machine m, it can be preceded (respectively succeeded) by at most one job-operation from the set of operations that the machine is capable of processing. Note that if (k, l) is the first operation to be processed on this machine, it will not be preceded by any other operation; and likewise if (k, l) is the last operation to be processed, it will not be succeeded by any other operation. In both of these cases, the left-hand sides of (3.6) and (3.7) will be zero, which trivially yield valid relationships. Also, if (k, l) is not assigned to machine m, then all the direct precedence $\boldsymbol{y}$-variables that relate (k, l) to other operations on machine m are validly set equal to zero by (3.6) and (3.7). Constraints (3.8) guarantee that if a machine has some $\sum_{(i,j)\in Z_m} x^m_{(i,j)}$ operations assigned to it for processing, then there must exist one less than this number of direct precedence variables that are set equal to 1 for this machine. These constraints are written as inequalities rather than as equalities to account for the case where the number of operations assigned to a machine is actually zero. Also, together with (3.6) and (3.7), these constraints establish the definitional role of the $\boldsymbol{y}$-variables. Constraints (3.9) enforce budgetary restrictions on each job i under every processing time scenario s. These constraints permit the sum of processing costs and travel costs for all operations of a job to exceed the budget by an amount of Δ^s_i, but with a corresponding penalty in the objective function. Note that the travel cost for each job i is assumed to be proportional to the number of parts, w_i, in that job. Constraints (3.10) enforce the relationship between the $\boldsymbol{x}$- and $\boldsymbol{v}$-variables according to $v^{(e,f)}_{(i,j,j+1)} = x^e_{(i,j)} x^f_{(i,j+1)}$ using a standard linearization technique whereby $v^{(e,f)}_{(i,j,j+1)} = 1$ if and only if both $x^e_{(i,j)} = 1$ and $x^f_{(i,j+1)} = 1$. Note that the $\boldsymbol{v}$-variables account for the required transfer between the machines in the objective function (3.1) and in Constraints (3.2) and (3.9). As such, because of the positive coefficients associated with these variables in the objective function (3.1) and the less-than-or-equal-to ($\leq$) relationships in (3.2) and (3.9), we could omit the first two sets of $\leq$ restrictions in (3.10) and have them automatically hold true at optimality. Constraints (3.11), (3.12), (3.13), and (3.14) ascribe nonnegativity and binary restrictions on the decision variables, while the $\boldsymbol{v}$-variables will automatically turn out to be binary-valued even though declared to be continuous on [0, 1]. Note also that the nonnegativity on the $\boldsymbol{t}$-variables is implied by (3.2), (3.3), (3.12), and (3.13).

The value of $H^{(m,s)}_{(i,j,k,l)}$ used in (3.4) can be prescribed as follows. Note that, if $y^m_{(i,j,k,l)} = 0$, then this constraint reduces to

$$t^s_{(i,j)} + p^{(m,s)}_{(k,l)} + u^m_{(i,j,k,l)} - t^s_{(k,l)} \leq H^{(m,s)}_{(i,j,k,l)}. \tag{3.15}$$

Hence, it is sufficient to assign to $H^{(m,s)}_{(i,j,k,l)}$ a valid upper bound on the left-hand side expression in (3.15). Given conservative bounds for $t^s_{(i,j)}$ such that $\left(t^s_{(i,j)}\right)_{\min} \leq t^s_{(i,j)} \leq \left(t^s_{(i,j)}\right)_{\max}$, we can set

$$H^{(m,s)}_{(i,j,k,l)} = \left(t^s_{(i,j)}\right)_{\max} + p^{(m,s)}_{(k,l)} + u^m_{(i,j,k,l)} - \left(t^s_{(k,l)}\right)_{\min}. \tag{3.16}$$

With respect to the bounds for $t^s_{(i,j)}$, we take

$$\left(t^s_{(i,j)}\right)_{\min} = r_i + \sum_{j' \leq j} \min_{m \in M(i,j')} \left\{p^{(m,s)}_{(i,j')}\right\} + \sum_{2 \leq j' \leq j} \min_{\substack{e \in M(i,j'-1) \\ f \in M(i,j')}} \{d_{(e,f)}\},$$

$$\left(t^s_{(i,j)}\right)_{\max} = \tau^s - \sum_{j' > j} \min_{m \in M(i,j')} \left\{p^{(m,s)}_{(i,j')}\right\} - \sum_{j < j' \leq J_i} \min_{\substack{e \in M(i,j'-1) \\ f \in M(i,j')}} \{d_{(e,f)}\},$$

$$\forall i = 1, \ldots, N, j = 1, \ldots, J_i, s = 1, \ldots, S, \tag{3.17}$$

where τ^s is some conservative upper bound on the overall makespan of all the jobs under scenario s. We used the value of τ^s to be the sum of the processing times of all the operations of the jobs.

3.2.2 The L-Shaped Method for the NFJS Problem

Formulation NFJSP can be decomposed into the following Stage-I (master) and Stage-II (recourse) problems:

Stage-I: Master Problem
MP: Minimize

$$\begin{aligned} &\sum_{s=1}^{S} \phi_s \sum_{i=1}^{N} \sum_{j=1}^{J_i} \sum_{m \in M_{(i,j)}} p^{(m,s)}_{(i,j)} x^m_{(i,j)} + \sum_{m \in M} \sum_{(i,j) \in Z_m} \sum_{\substack{(k,l) \in Z_m \\ (i,j) \neq (k,l)}} y^m_{(i,j,k,l)} u^m_{(i,j,k,l)} \\ &+ \sum_{i=1}^{N} \sum_{j=1}^{J_i - 1} \sum_{e \in M_{(i,j)}} \sum_{f \in M_{(i,j+1)}} v^{(e,f)}_{(i,j,j+1)} d_{(e,f)} + \sum_{s=1}^{S} \phi_s \mathbf{Q}(\mathbf{x}, \mathbf{y}, \mathbf{v}, s) \end{aligned} \tag{3.18}$$

subject to: (3.5), (3.6), (3.7), (3.8), (3.10), (3.12), (3.13), and (3.14), where $\mathbf{Q}(\mathbf{x}, \mathbf{y}, \mathbf{v}, s)$ is the recourse function corresponding to the optimal value of the subproblem that minimizes the penalized sum of job completion times and budget over-runs for a given assignment vector $\mathbf{x}$, sequencing vector $\mathbf{y}$, tracking vector $\mathbf{v}$, and for a processing time scenario s. The linear recourse subproblem for scenario s is given by:

Stage-II: Recourse Problem

$$\mathbf{RP}: \quad \mathbf{Q}(\mathbf{x}, \mathbf{y}, \mathbf{v}, s) = \text{Min} \sum_{i=1}^{N} \alpha_i t^s_{(i,J_i)} + \sum_{i=1}^{N} \beta_i \Delta^s_i \tag{3.19}$$

subject to: (3.2), (3.3), (3.4), (3.9), and (3.11).

We note that, in the decomposition outlined above for formulation NFJSP, the master problem could generate an assignment and sequencing solution that might not be feasible to the subproblem. There are three possible causes for such infeasibility. First, the Stage-I formulation does not exclude "subtours" while sequencing operations assigned to a particular machine. For example, suppose that operations (a, b), (c, d), and (e, f) are assigned to machine m, so that

$$x^m_{(a,b)} = x^m_{(c,d)} = x^m_{(e,f)} = 1.$$

One can verify that the following values of direct precedence variables are feasible to the Stage-I formulation (in particular, satisfies Constraint (3.8)):

$$\begin{aligned} y^m_{(a,b,c,d)} &= 1;\ y^m_{(a,b,e,f)} = 0; \\ y^m_{(c,d,a,b)} &= 1;\ y^m_{(c,d,e,f)} = 0; \\ y^m_{(e,f,a,b)} &= 0;\ y^m_{(e,f,c,d)} = 0. \end{aligned}$$

However, due to the subtour between (a, b) and (c, d), this solution does not represent a valid processing sequence. In the NFJSP formulation, this kind of subtour is eliminated by Constraints (3.4), which are not included in the master problem. Second, note that Constraints (3.2) in NFJSP, upon decomposition, become part of the subproblem and capture the fact that the completion time of a lower indexed operation of a job must be less than or equal to that for any higher indexed operations of the same job. In NFJSP, Constraints (3.2) in conjunction with other constraints that determine the value of the $\boldsymbol{y}$-variables (Constraints (3.6), (3.7), and (3.8)) ensure that, in the case of re-entrant flow, where a job visits a machine for multiple operations, the lower indexed operations of a job are sequenced before a higher indexed operation of the same job. However, since Constraints (3.2) are

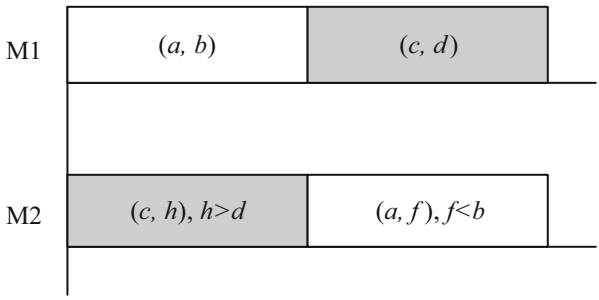


Fig. 3.1 A deadlock configuration involving two machines

no longer a part of the master problem, its absence may result in an assignment and sequencing vector that does not honor the re-entrant flow conditions. Third, the assignment and sequencing vectors from the master problem may cause a *deadlock*. This occurs in the face of a certain configuration of assignment and sequencing decisions that result in a circuit or a cycle wherein each operation in the cycle waits for another operation within the cycle to complete processing. This is illustrated in Fig. 3.1. Note that on machine M1, (c, d) waits for (a, b) to finish processing according to the sequencing decision. Operation (a, b) on machine M1 must follow (a, f) on machine M2 owing to operating precedence constraints. However, on machine M2, operation (a, f) follows (c, h), which, in turn, can begin only after (c, d) on machine M1 has been completed. Thus, none of the four operations can begin processing, resulting in a deadlock.

As a result of this potential infeasibility, the above decomposition of NFJS problem does not possess the property of relatively complete recourse. In order to render the first-stage solution feasible to the second-stage, it is necessary to obviate the infeasibility due to subtours, re-entrant flows, and deadlocks. One way to achieve this is through the use of artificial variables as described by van Slyke and Wets (1969). These variables are inserted into the subproblems for every scenario and feasibility cuts are developed that become a part of the master problem, which in turn ultimately induce the master problem to generate solutions that are feasible to the subproblems. Accordingly, for a given output (**x**, **y, v**) from the master problem, the following augmented recourse problem (ARP) is solved, one for each scenario s:

ARP: Minimize

$$\sum_{i=1}^{N}\sum_{j=1}^{J_i} a^{s}_{1(i,j)} + \sum_{m\in M}\sum_{(i,j)\in Z_m}\sum_{\substack{(k,l)\in Z_m \\ (k,l)\neq(i,j)}} a^{(m,s)}_{2(i,j,k,l)} \tag{3.20}$$

subject to:

$$t^s_{(i,j)} + \sum_{m\in M_{(i,j+1)}} \left(p^{(m,s)}_{(i,j+1)} x^m_{(i,j+1)}\right) + \sum_{e\in M_{(i,j)}} \sum_{f\in M_{(i,j+1)}} d_{(e,f)} v^{(e,f)}_{(i,j,j+1)} - a^s_{1(i,j+1)} \le t^s_{(i,j+1)},$$
$$\forall i = 1,\ldots,N, j = 1,\ldots,J_i - 1 \tag{3.21}$$

$$r_i + \sum_{m\in M_{(i,1)}} \left(p^{(m,s)}_{(i,1)} x^m_{(i,1)}\right) - a^s_{1(i,1)} \le t^s_{(i,1)}, \quad \forall i = 1,\ldots,N \tag{3.22}$$

$$t^s_{(i,j)} + p^{(m,s)}_{(k,l)} + u^m_{(i,j,k,l)} - a^{(m,s)}_{2(i,j,k,l)} \le t^s_{(k,l)} + \left(1 - y^m_{(i,j,k,l)}\right) H^{(m,s)}_{(i,j,k,l)},$$
$$\forall m \in M, \forall\ (i,j) \ne (k,l)\ \text{ in } Z_m \tag{3.23}$$

$$\sum_{j=1}^{J_i} \sum_{m\in M_{(i,j)}} c^m_{(i,j)} p^{(m,s)}_{(i,j)} x^m_{(i,j)} + \left(\sum_{j=1}^{J_i-1} \sum_{e\in M_{(i,j)}} \sum_{f\in M_{(i,j+1)}} q_{(e,f)} v^{(e,f)}_{(i,j,j+1)}\right) w_i - \Delta^s_i \le b_i,$$
$$\forall i = 1,\ldots,N \tag{3.24}$$

$$a^s_{1(i,j)} \ge 0, \quad \forall i = 1,\ldots,N, j = 1,\ldots,J_i \tag{3.25}$$

$$a^{(m,s)}_{2(i,j,k,l)} \ge 0, \quad \forall m \in M, \forall\ (i,j) \ne (k,l) \in Z_m \tag{3.26}$$

$$t^s_{(i,j)} \ge 0, \quad \forall i = 1,\ldots,N, j = 1,\ldots,J_i \tag{3.27}$$

$$\Delta^s_i \ge 0, \quad \forall i = 1,\ldots,N. \tag{3.28}$$

Note that artificial variables are included in Constraints (3.2), (3.3), and (3.4), which now become (3.21), (3.22), and (3.23), respectively. Constraints (3.9) do not require any artificial variables because they can always be satisfied by virtue of the budget over-run variables Δ^s_i, $i = 1, \ldots, N$. Whereas the corresponding restrictions are included in (3.24), they can be effectively omitted from Problem ARP.

If the value of the objective function (3.20) in ARP equals zero for all the subproblems, then it indicates that the solution from the master program (first-stage) is feasible to the recourse (second-stage) problem. However, if there exists a scenario, say $\overline{s}$, such that the subproblem corresponding to this scenario has a positive optimal objective value, then a feasibility cut is generated so as to eliminate the corresponding solution from the master program, as follows. Rewriting Constraints (3.21), (3.22), and (3.23) as "$\ge$" inequalities, we associate nonnegative dual variables $\eta^{\overline{s}}_{(i,j)}$, $\eta^{\overline{s}}_i$, $\eta^{(m,\overline{s})}_{(i,j,k,l)}$ with these respective constraints. Note that (3.24) has been dropped from Problem ARP. Then, we derive the following feasibility cut for scenario $\overline{s}$:

$$\sum_{i=1}^{N}\sum_{j=1}^{J_i-1}\eta_{(i,j)}^{\bar{s}}\left(\sum_{m\in M_{(i,j+1)}}p_{(i,j+1)}^{(m,\bar{s})}x_{(i,j+1)}^{m}+\sum_{e\in M_{(i,j)}}\sum_{f\in M_{(i,j+1)}}d_{(e,f)}v_{(i,j,j+1)}^{(e,f)}\right)$$
$$+\sum_{i=1}^{N}\eta_{i}^{\bar{s}}\sum_{m\in M_{(i,1)}}p_{(i,1)}^{(m,\bar{s})}x_{(i,1)}^{m}+\sum_{m\in M}\sum_{(i,j)\in Z_m}\sum_{\substack{(k,l)\in Z_m\\(k,l)\neq(i,j)}}\eta_{(i,j,k,l)}^{(m,\bar{s})}H_{(i,j,k,l)}^{(m,\bar{s})}y_{(i,j,k,l)}^{m}$$
$$\leq-\sum_{i=1}^{N}\eta_{i}^{\bar{s}}r_i+\sum_{m\in M}\sum_{(i,j)\in Z_m}\sum_{\substack{(k,l)\in Z_m\\(k,l)\neq(i,j)}}\eta_{(i,j,k,l)}^{(m,\bar{s})}\left(H_{(i,j,k,l)}^{(m,\bar{s})}-p_{(k,l)}^{(m,\bar{s})}-u_{(i,j,k,l)}^{m}\right). \tag{3.29}$$

This feasibility cut is appended to the master program. Whenever the objective function values for all the augmented subproblems equal zero, the Stage-I solution yields feasible Stage-II recourse problems, whence we either verify optimality or generate optimality cuts as described next.

3.2.3 Optimality Cuts

When a Stage-I solution $(\overline{\mathbf{x}},\overline{\mathbf{y}},\overline{\mathbf{v}})$ is feasible for the separable Stage-II problems, the latter effectively determine optimal values for the completion time and budget over-run variables for each scenario. This yields the expected recourse value of the Stage-II objective function as given by $\mathbf{Q}(\overline{\mathbf{x}},\overline{\mathbf{y}},\overline{\mathbf{v}}) \equiv \sum_{s=1}^{S}\phi_s\mathbf{Q}(\overline{\mathbf{x}},\overline{\mathbf{y}},\overline{\mathbf{v}},s)$. This value is then compared with the lower bound ($\overline{\theta}$, say) on the recourse value as previously obtained by solving the master problem. Note that (3.18) evaluated for $(\overline{\mathbf{x}},\overline{\mathbf{y}},\overline{\mathbf{v}})$ provides an upper bound for the NFJS problem given the feasibility of $(\overline{\mathbf{x}},\overline{\mathbf{y}},\overline{\mathbf{v}})$, and can be used to update the incumbent objective function value. If $\overline{\theta}\geq\mathbf{Q}(\overline{\mathbf{x}},\overline{\mathbf{y}},\overline{\mathbf{v}})$, we have that $(\overline{\mathbf{x}},\overline{\mathbf{y}},\overline{\mathbf{v}})$ is an optimal solution to the NFJS problem. Otherwise, if $\overline{\theta}<\mathbf{Q}(\overline{\mathbf{x}},\overline{\mathbf{y}},\overline{\mathbf{v}})$, we generate an optimality cut to help close the gap between the two bounds. Letting $\xi_{(i,j)}^{s}$, ξ_{i}^{s}, $\xi_{(i,j,k,l)}^{(m,s)}$, and ω_{i}^{s} be the nonnegative dual variables associated with respect to Constraints (3.2), (3.3), (3.4), and (3.9) written as $\geq$ restrictions, the optimality cut is given as follows, where, as mentioned above, θ is used to represent the final term in the objective function (3.18) of the master program:

$$\theta \geq \sum_{s=1}^{S} \phi_s \left\{ \sum_{i=1}^{N} \sum_{j=1}^{J_i-1} \xi_{(i,j)}^{s} \left(\sum_{m \in M_{(i,j+1)}} p_{(i,j+1)}^{(m,s)} x_{(i,j+1)}^{m} + \sum_{e \in M_{(i,j)}} \sum_{f \in M_{(i,j+1)}} d_{(e,f)} v_{(i,j,j+1)}^{(e,f)} \right) \right.$$

$$+ \sum_{i=1}^{N} \xi_i^s \sum_{m \in M_{(i,1)}} p_{(i,1)}^{(m,s)} x_{(i,1)}^{m} + \sum_{m \in M} \sum_{(i,j) \in Z_m} \sum_{\substack{(k,l) \in Z_m \\ (k,l) \neq (i,j)}} \xi_{(i,j,k,l)}^{(m,s)} H_{(i,j,k,l)}^{(m,s)} y_{(i,j,k,l)}^{m}$$

$$+ \sum_{i=1}^{N} \omega_i^s \left[\sum_{j=1}^{J_i} \sum_{m \in M_{(i,j)}} c_{(i,j)}^{m} p_{(i,j)}^{(m,s)} x_{(i,j)}^{m} + \left(\sum_{j=1}^{J_i-1} \sum_{e \in M_{(i,j)}} \sum_{f \in M_{(i,j+1)}} q_{(e,f)} v_{(i,j,j+1)}^{(e,f)} \right) w_i \right]$$

$$+ \sum_{i=1}^{N} \xi_i^s r_i - \sum_{m \in M} \sum_{(i,j) \in Z_m} \sum_{\substack{(k,l) \in Z_m \\ (k,l) \neq (i,j)}}$$

$$\left. \xi_{(i,j,k,l)}^{(m,s)} \left(H_{(i,j,k,l)}^{(m,s)} - p_{(k,l)}^{(m,s)} - u_{(i,j,k,l)}^{m} \right) - \sum_{i=1}^{N} \omega_i^s b_i. \right. \tag{3.30}$$

The optimality cut is appended to the MP and the revised MP is re-solved. The iterations continue in this fashion until the lower and upper bounds converge (or come within a desired optimality tolerance).

Note that the master problem and the linear programs corresponding to the subproblems need to be re-solved every time a new feasibility or optimality cut is added. This can lead to a lengthy process in case a large number of feasibility cuts are required to generate a feasible solution. Therefore, it is helpful to a priori include suitable valid inequalities in the master problem to induce second-stage feasibility. We present such inequalities next.

3.2.4 Alternative Valid Inequalities for Inducing Stage-II Feasibility That Also Provide a Stage-I Lower Bound

The alternative set of valid inequalities derived in this section relies on the fact that for any fixed value of (**x**, **y**, **v**), the feasibility of the Stage-II problem does not depend on a particular scenario. In other words, if the routing and sequencing decisions are feasible for a given scenario, then they are also feasible for any other scenario, because changes in job processing times can be accommodated by adjusting completion times while maintaining the feasibility of the subproblem constraints. Consequently, variables and constraints of the subproblem for a given scenario can be included in the master problem to induce feasibility of the

subproblems for all the scenarios. The next result indicates that the particular scenario that use the expected values of processing times provides a lower bound on $\theta \equiv \mathbf{Q}(\mathbf{x}, \mathbf{y}, \mathbf{v})$.

Proposition 1 The optimal objective value of the subproblem with expected processing times yields a lower bound on $\mathbf{Q}(\mathbf{x}, \mathbf{y}, \mathbf{v}) \equiv \sum_{s=1}^{S} \phi_s \mathbf{Q}(\mathbf{x}, \mathbf{y}, \mathbf{v}, s)$.

Proof For any fixed values of $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{v}$, the recourse function $\mathbf{Q}(\mathbf{x}, \mathbf{y}, \mathbf{v}, s)$ is only a function of s. We rewrite this as $\mathbf{Q}(\boldsymbol{p}^s)$, where $\boldsymbol{p}^s$ is the vector of operation processing times.

Let $\boldsymbol{p}^E = \sum_{s=1}^{S} \phi_s \boldsymbol{p}^s$. The fact that $\mathbf{Q}\left(\boldsymbol{p}^E\right) = \mathbf{Q}\left(\sum_{s=1}^{S} \phi_s \boldsymbol{p}^s\right) \leq \sum_{s=1}^{S} \phi_s \mathbf{Q}\left(\boldsymbol{p}^s\right)$ is easily established because $\mathbf{Q}(\boldsymbol{p}^s)$ is a convex function of $\boldsymbol{p}^s$, due to fixed recourse (see Theorem 5 in Birge and Louveaux 2000, p. 89.) □

Accordingly, we define the following variables:

$t^E_{(i,j)}$ = completion time of operation j of job i under expected processing times.
Δ^E_i = budget over-run for job i under expected processing times.

Then, by Proposition 1 and Constraints (3.2), (3.3), (3.4), and (3.9), we include the following set of restrictions in the Stage-I master program:

$$\theta \geq \sum_{i=1}^{N} \alpha_i t^E_{(i,J_i)} + \sum_{i=1}^{N} \beta_i \Delta^E_i \tag{3.31}$$

$$t^E_{(i,j)} + \sum_{m \in M_{(i,j+1)}} \left(p^{(m,E)}_{(i,j+1)} x^m_{(i,j+1)}\right) + \sum_{e \in M_{(i,j)}} \sum_{f \in M_{(i,j+1)}} d_{(e,f)} v^{(e,f)}_{(i,j,j+1)} \leq t^E_{(i,j+1)},$$
$$\forall i = 1, \ldots, N, \;\; j = 1, \ldots, J_i - 1 \tag{3.32}$$

$$r_i + \sum_{m \in M_{(i,1)}} \left(p^{(m,E)}_{(i,1)} x^m_{(i,1)}\right) \leq t^E_{(i,1)}, \quad \forall i = 1, \ldots, N \tag{3.33}$$

$$t^E_{(i,j)} + p^{(m,E)}_{(k,l)} + u^m_{(i,j,k,l)} \leq t^E_{(k,l)} + \left(1 - y^m_{(i,j,k,l)}\right) H^{(m,E)}_{(i,j,k,l)},$$
$$\forall m \in M, \forall\, (i,j) \neq (k,l) \in Z_m \tag{3.34}$$

$$\sum_{j=1}^{J_i} \sum_{m \in M_{(i,j)}} c^m_{(i,j)} p^{(m,E)}_{(i,j)} x^m_{(i,j)} + \left(\sum_{j=1}^{J_i - 1} \sum_{e \in M_{(i,j)}} \sum_{f \in M_{(i,j+1)}} q_{(e,f)} v^{(e,f)}_{(i,j,j+1)}\right) w_i - \Delta^E_i \leq b_i,$$
$$\forall i = 1, \ldots, N \tag{3.35}$$

$$t^E_{(i,j)} \geq 0, t^E_{(i,j)} \geq 0, \quad \forall i = 1, \ldots, N, j = 1, \ldots, J_i \tag{3.36}$$

$$\Delta^E_i \geq 0, \quad \forall i = 1, \ldots, N. \tag{3.37}$$

Note that $p^{(m,E)}_{(i,j)} \equiv \sum_{s=1}^{S} \phi_s p^{(m,s)}_{(i,j)}$, and that $H^{(m,E)}_{(i,j,k,l)}$ is pre-calculated similar to that in (3.16) and (3.17), with $p^{(m,E)}_{(i,j)}$ replacing $p^{(m,s)}_{(i,j)}$ in the expressions.

Also note that the infeasibility caused by re-entrant flow will be eliminated by Constraints (3.32), (3.33), and (3.34), since they enforce a proper ordering (via the $t^{E}_{(i,j)}$ -variables) among the operations of the same job. These constraints also prevent the occurrence of a deadlock as follows. Consider the situation depicted in Fig. 3.1, where we have $x^{M1}_{(a,b)} = x^{M1}_{(c,d)} = x^{M2}_{(c,h)} = x^{M2}_{(a,f)} = 1$ and $y^{M1}_{(a,b,c,d)} = y^{M2}_{(c,h,a,f)} = 1$. Since $h > d$ and $f < b$, the constraint set (3.32) asserts that $t^{E}_{(c,h)} > t^{E}_{(c,d)}$ and $t^{E}_{(a,f)} < t^{E}_{(a,b)}$. On the other hand, the constraint set (3.34) enforces $t^{E}_{(a,b)} < t^{E}_{(c,d)}$ and $t^{E}_{(c,h)} < t^{E}_{(a,f)}$. Clearly, these four inequalities lead to a contradiction, and consequently, the corresponding values of the **x**- and **y**-variables would be infeasible to the master problem augmented with Constraints (3.32) and (3.34).

Note that (3.34) also serves to eliminate subtours among the operations processed on a machine. These are essentially the MTZ-type of subtour elimination constraints (Miller et al. 1960), and they can be weak in the sense that they lead to loose LP relaxations. However, they can potentially be strengthened through the use of flow-based valid inequalities as shown by Sherali et al. (2006).

3.3 Valid Inequalities for Further Tightening the Model Formulation

In this section, we develop three classes of valid inequalities by exploiting the inherent structure of the NFJS problem. These inequalities are added to the MP to tighten its continuous relaxation and provide better lower bounds for use in the branch-and-bound algorithm for solving NFJSP. The first type of inequalities arises from the flow balance-type constraints that capture the movement of operations among the machines. The other two types of inequalities are formulated to obviate infeasibility caused by re-entrant flow and deadlock, respectively. They are based on a new formulation for the asymmetric travelling salesman problem (ATSP) presented in Sarin et al. (2005).

3.3.1 Flow-Balance Constraints

In the NFJSP formulation, we used the variable $v^{(e,f)}_{(i,j,j+1)}$ to represent the transfer of job i from machine e to machine f when performing the respective operations j and $j+1$. This definitional role of $v^{(e,f)}_{(i,j,j+1)}$ is enforced by (3.10). We can further tighten the continuous relaxation of the model by introducing the following flow-balance constraints (FBC):

$$\sum_{f\in M_{(i,2)}} v^{(e,f)}_{(i,1,2)} = x^{e}_{(i,1)}, \quad \forall i = 1,\ldots,N, e \in M_{(i,1)} \tag{3.38}$$

$$\sum_{e\in M_{(i,j-1)}} v^{(e,f)}_{(i,j-1,j)} = \sum_{g\in M_{(i,j+1)}} v^{(f,g)}_{(i,j,j+1)}, \quad \forall i = 1,\ldots,N, j = 2,\ldots,J_i - 1, f \in M_{(i,j)} \tag{3.39}$$

$$\sum_{e\in M_{(i,J_i-1)}} v^{(e,f)}_{(i,J_i-1,J_i)} = x^{f}_{(i,J_i)}, \quad \forall i = 1,\ldots,N, f \in M_{(i,J_i)}. \tag{3.40}$$

Constraints (3.38) assert that if the first operation of job i is assigned to machine e, then job i must be transported from machine e to some machine f that is capable of processing the second operation of job i. Constraints (3.39) capture the fact that if machine f is chosen for processing the jth operation of job i, $1 < j < J_i$, then job i is necessarily transferred from some previous machine, e, and is transported to a succeeding machine, g, while performing the respective operations $j-1$ and $j+1$. Similarly, Constraints (3.40) require job i to be transferred from some previous machine e in case its last operation is processed on machine f.

3.3.2 *Re-Entrant Flow-Based Constraints*

In the case of re-entrant flows, the lower indexed operations of any job must precede the higher indexed operations of that job for the sequence to be feasible. For the sake of convenience, we designate an order $ord(i, j)$ for elements of Z_m, $\forall m \in M$, such that the ordering of operations from the same job is maintained. For instance, if $Z_m = \{(1, 1), (2, 2), (1, 2)\}$, we can assign $ord(1, 1) = 1$, $ord(2, 2) = 2$, and $ord(1, 2) = 3$. Based on this definition, we let

$$h^{m}_{(i,j,k,l)} \equiv x^{m}_{(i,j)} x^{m}_{(k,l)}, \forall m \in M; (i,j), (k,l) \in Z_m : ord(i,j) < ord(k,l)$$

We can linearize the foregoing relationship between the **h**- and the **x**-variables by using the following logical constraints:

$$h^{m}_{(i,j,k,l)} \le x^{m}_{(i,j)}, \quad h^{m}_{(i,j,k,l)} \le x^{m}_{(k,l)}, \quad h^{m}_{(i,j,k,l)} \ge x^{m}_{(i,j)} + x^{m}_{(k,l)} - 1,$$
$$\forall m \in M, (i,j), (k,l) \in Z_m : ord(i,j) < ord(k,l). \tag{3.41}$$

We also define certain *indirect precedence variables* as follows:

$$g^{m}_{(i,j,k,l)} = \begin{cases} 1, & \text{if operation } j \text{ of job } i \text{ is processed sometime before} \\ & \text{operation } l \text{ of job } k \text{ on machine } m, \\ 0, & \text{otherwise.} \end{cases}$$

Then, we have,

$$g^m_{(i,j,k,l)} + g^m_{(k,l,i,j)} = h^m_{(i,j,k,l)}, \quad \forall m \in M, (i,j), (k,l) \in Z_m : ord(i,j) < ord(k,l) \tag{3.42}$$

$$g^m_{(i,j,\gamma,\eta)} \geq g^m_{(i,j,k,l)} + g^m_{(k,l,\gamma,\eta)} - 1, \quad \forall m \in M, \forall \text{distinct } (i,j), (k,l), (\gamma,\eta) \in Z_m \tag{3.43}$$

$$g^m_{(i,j,i,l)} = 0, \quad \forall m \in M, (i,j), (i,l) \in Z_m : j > l. \tag{3.44}$$

Constraints (3.42) state that given two job-operations on a machine, one of them must either precede or succeed the other. Constraints (3.43) represent the transitivity property; that is, for any triplet of job-operations (i,j), (k,l), and (γ,η) on machine m, if operation (i,j) is scheduled somewhere before operation (k,l) and operation (k,l) is scheduled somewhere before operation (γ,η), then operation (i,j) must necessarily be scheduled before operation (γ,η). Finally, the re-entrant flow Constraints (3.44) ensure that if two operations of the same job are assigned to a machine, then the lower indexed job operation precedes the higher indexed operation.

Also, we have the following logical constraints connecting the indirect and the direct precedence variables:

$$g^m_{(i,j,k,l)} \geq y^m_{(i,j,k,l)}, \quad \forall m \in M, \forall (i,j) \neq (k,l) \text{ in } Z_m. \tag{3.45}$$

Hence, the re-entrant flow constraints that can be accommodated into the master (Stage-I) program are given by (3.41), (3.42), (3.43), (3.44), and (3.45). Note that by introducing the $g^m_{(i,j,k,l)}$ -variables, we also eliminate infeasibility caused by subtours in operation sequencing, since the indirect precedence enforced by the $g^m_{(i,j,k,l)}$ -variables precludes the occurrence of subtours.

3.3.3 Deadlock Prevention Constraints

Next, we develop valid inequalities to prevent the occurrence of a deadlock. For the sake of brevity, we only present inequalities for the prevention of 2-machine deadlocks and establish their validity. For a detailed development of the corresponding results for the general case of m-machine deadlocks, see Varadarajan (2006).

Consider the following situation in a job shop environment: operations (a, b) and (c, d) are assigned to machine m; and operations (a, f) and (c, h), where $f < b$ and $h > d$, are assigned to machine n. If (a, b) precedes (c, d), then (a, f) must necessarily

precede (c, h) to avoid a deadlock (see Fig. 3.1, where machines m and n are denoted by M1 and M2, respectively). Then, we have the following result:

Proposition 2 Two-machine deadlocks are prevented by including the following additional inequalities:

$$\begin{aligned} & g^n_{(a,f,c,h)} \geq g^m_{(a,b,c,d)} + h^n_{(a,f,c,h)} - 1, \\ & \quad \forall\, m, n \in M, (a,b), (c,d) \in Z_m, (a,f), (c,h) \in Z_n, \\ & \quad f < b \text{ and } h > d. \end{aligned} \tag{3.46}$$

Proof If $h^n_{(a,f,c,h)} = 0$, then $g^n_{(a,f,c,h)} = 0$ by (3.42), and $g^m_{(a,b,c,d)}$ can be 1 without causing any deadlock. If $h^n_{(a,f,c,h)} = 1$ and $g^m_{(a,b,c,d)} = 0$, then $g^n_{(a,f,c,h)} \geq 0$, and the schedule is deadlock free. On the other hand, if both $g^m_{(a,b,c,d)}$ and $h^n_{(a,f,c,h)}$ are equal to 1, then (a, b) is processed sometime before (c, d) on machine m, and (a, f) and (c, h) are processed on the same machine n. To yield a deadlock-free schedule under this situation, (a, f) must be processed sometime before (c, h) on machine n, which is enforced by (3.46). □

Note that to apply the above deadlock prevention constraints to the master problem, we need to also include Constraints (3.41), (3.42), (3.43), and (3.44), so that $\boldsymbol{g}$- and $\boldsymbol{h}$-variables take their definitional roles in the model.

3.4 Computational Results

We now present results of computational experimentation to demonstrate the effectiveness of our feasibility-inducing and model-tightening inequalities within the framework of the L-shaped method for the solution of NFJS problem. In this method, the Stage-I master problem is solved using a branch-and-bound algorithm. Whenever an integer solution is obtained for a node problem's LP relaxation, the Stage-II problem is solved to verify its feasibility and optimality. If any of these conditions are not met, a feasibility cut or an optimality cut is generated and added to the Stage-I problem, and the branch-and-bound process continues.

There are several ways in which the valid inequalities pertaining to the expected value scenario (EVS), the re-entrant flows (RF), and deadlock prevention (DP) (developed in Sects. 3.2.4, 3.3.2, and 3.3.3, respectively) can be applied. We can either use them separately, or we can apply the EVS inequalities in conjunction with selected members of the RF and DP inequalities in order to tighten the underlying relaxation. Our preliminary investigation has shown that the use of the EVS inequalities always leads to shorter CPU times. The question, then, is how (if at all) to apply the RF and DP inequalities in addition to the EVS inequalities. Note that, to achieve the full potential of the RF and DP inequalities, we need to consider re-entrant flows and deadlocks among all the machines, which would require a large number of extra variables and constraints that may overburden the master program and deteriorate its computational performance. Therefore, we choose to apply these

inequalities to a proper subset of machines, as investigated in Sect. 3.4.2 below. Furthermore, we explore the optional addition of the flow-balance constraints FBC of Sect. 3.3.1.

3.4.1 Design of Test Problems

To represent routing flexibility, we group machines into work centers; operations assigned to a work center are allowed to be processed by any machine in that work center. Due to this feature, the manner in which workload is assigned to the machines within a work center is not determined until a solution is obtained. To indicate the potential workload on a machine, we define a *load factor* to be the total number of visits of all the jobs to that machine on average, assuming that all the machines within a work center equally share the workload. According to the load factor, we differentiate machines into two categories: low-number-of-visit (LNV) machines (with two potential visits on average), or high-number-of-visit (HNV) machines (with three potential visits on average). Consequently, three job-visiting patterns are considered, pertaining to different distributions of workload on the machines. These are: {"$L^+ \bullet H^-$," "L•H," "$L^- \bullet H^+$"}. The letters "L" and "H" refer to the LNV and HNV machines, respectively; the plus/minus signs in the superscript indicate that, relatively, there are higher or lower number of machines in a category than those in the other. We consider test problems of various sizes, involving 6, 8, or 10 machines. Their basic specifications are listed in Table 3.1.

With respect to routing flexibility, three cases are considered. In Case ρ_1, all HNV machines are grouped into one work center, while no routing flexibility exists among the LNV machines. In Case ρ_2, all LNV machines are grouped into one work center; no routing flexibility exists among the HNV machines. In Case ρ_3, no routing flexibility exists.

Table 3.1 Specifications for various problem sizes

Number of machines	Job-visiting pattern	Number of LNV machines	Number of HNV machines	Number of jobs	Total number of operations
6	$L^+ \bullet H^-$	4	2	3	14
	L•H	3	3	3	15
	$L^- \bullet H^+$	2	4	3	16
8	$L^+ \bullet H^-$	5	3	4	22
	L•H	4	4	4	24
	$L^- \bullet H^+$	3	5	4	26
10	$L^+ \bullet H^-$	6	4	5	28
	L•H	5	5	5	30
	$L^- \bullet H^+$	4	6	5	32

For each of the above $3 \times 3 \times 3 = 27$ combinations of numbers of machines, job-visiting patterns, and routing flexibility, we constructed 20 test problems with randomly generated job routings and processing times. For the 6-machine problems, we considered the following numbers of scenarios: {100, 200, 300, 400}. The larger-sized problems (involving 8 and 10 machines) were solved using 400 scenarios to reveal the effectiveness of the proposed strategy.

All experimental runs were implemented using AMPL-CPLEX 10.1 and performed on a Pentium D 3.2 GHz CPU computer with 2 GB memory.

3.4.2 Experimental Results

We first compare the performance of solving NFJSP directly by AMPL-CPLEX (designated as **Method I**) with that of our decomposition approach (designated as **Method II**), which includes the EVS inequalities but not the RF and DP inequalities. Results of the L-shaped method without any additional inequalities, i.e., only with the standard feasibility and optimality cuts (3.29) and (3.30) (designated as **Method III**) are also provided for comparison. In addition, we considered the option of either adding or not adding the FBC inequalities of Sect. 3.3.1 to these three methods. The 6-machine problems were solved to optimality; the 8- and 10-machine problems were run until an integer solution was obtained within an optimality gap of 5 %. Since the superiority of Method II was observed in our preliminary study, we adopted the following approach to avoid excessive run times: Method II was used to solve the test problems first. Since there are two options (with or without the FBC inequalities), we record the CPU time as t_2' and t_2", respectively, for these options. Let $t_2 = \max\{t_2', t_2''\}$. For Methods I and III, we set an upper bound on the CPU time of $\max\{1.2 \times t_2, t_c\}$, where the value of t_c is 1500, 2000, and 2500 s, respectively, for the 6-, 8-, and 10-machine problems. The first term $(1.2 \times t_2)$ is used to provide a reasonable (20 %) margin to demonstrate the superiority of Method II over the other two methods. The second term (t_c) is included to ensure that the effectiveness of adding the FBC inequalities is not obscured by stopping prematurely.

The results obtained are presented in Table 3.2. Note that the inclusion of the FBC inequalities results in shorter CPU times for Methods I and II in most cases. Therefore, in the following discussion, we only provide results for the case where the FBC inequalities have been added to the NFJSP formulation. From these results, it is also evident that Method II is substantially faster than Methods I and III.

To further illustrate how this dominance varies with an increasing number of scenarios, we present, in Fig. 3.2, the results for 6-machine problems with four different numbers of scenarios, namely, 100, 200, 300, and 400. The arrow and the number next to a data point indicate the percentage of test problems that consume CPU times more than the limit of 1500 s. Note that Method II substantially dominates the other methods as the number of scenarios increases from 100 to 400. This pattern is observed not only for the average values that are depicted in

Table 3.2 Computational results for Methods I, II, and III for 400 scenarios

	Average CPU time (in seconds)					
	Method I		Method II		Method III	
Number of machines	w/o FBC	w/ FBC	w/o FBC	w/ FBC	w/o FBC	w/ FBC
6	207.53	196.26	21.38	22.41	266.82	270.63
8	742.29	707.98	27.42	25.09	1365.04	1379.96
10[a]	1315.04	1306.59	82.81	54.63	2244.51	2242.22
10[b]	–	–	123.55	109.87	2273.06	2271.03

[a]Twenty-one (out of 180) 10-machine problems could not be solved by Method I due to excessive memory requirements. The average values in this row are calculated based on the remaining 159 test problems

[b]One (out of the 180) 10-machine problem could not be solved by Method III due to excessive memory requirements. The average values in this row are calculated based on the other 179 test problems

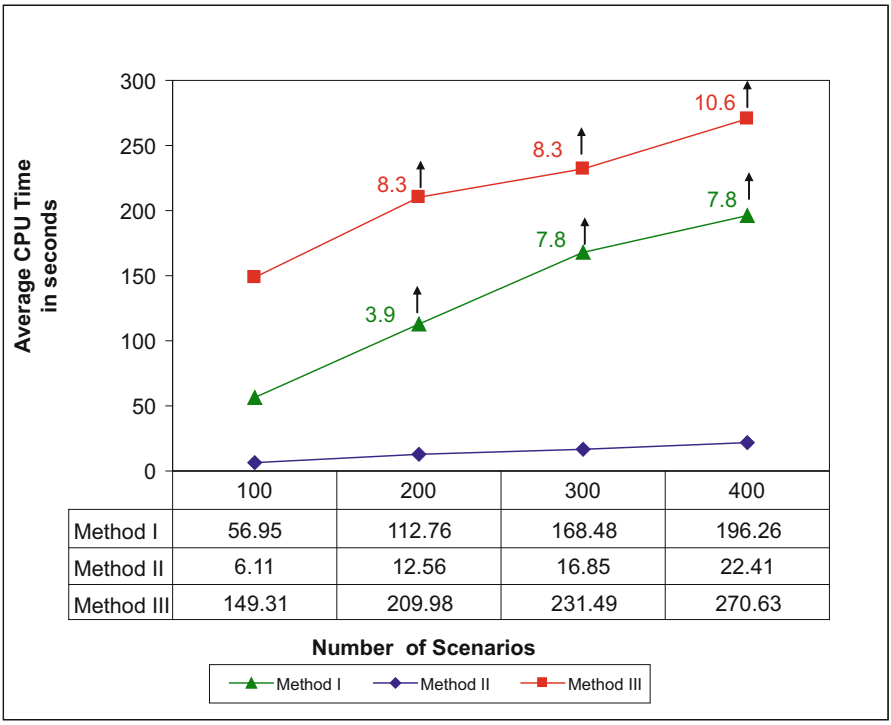


Fig. 3.2 Average CPU times for 6-machine problems for various numbers of scenarios

Fig. 3.2, but also for all combinations of job-visiting patterns and cases of routing flexibility. For the sake of brevity, we illustrate this behavior in Fig. 3.3 by using the combination "L•H" × "ρ_2". Clearly, Method II dominates the other two methods for all numbers of scenarios considered. Moreover, as the number of scenarios

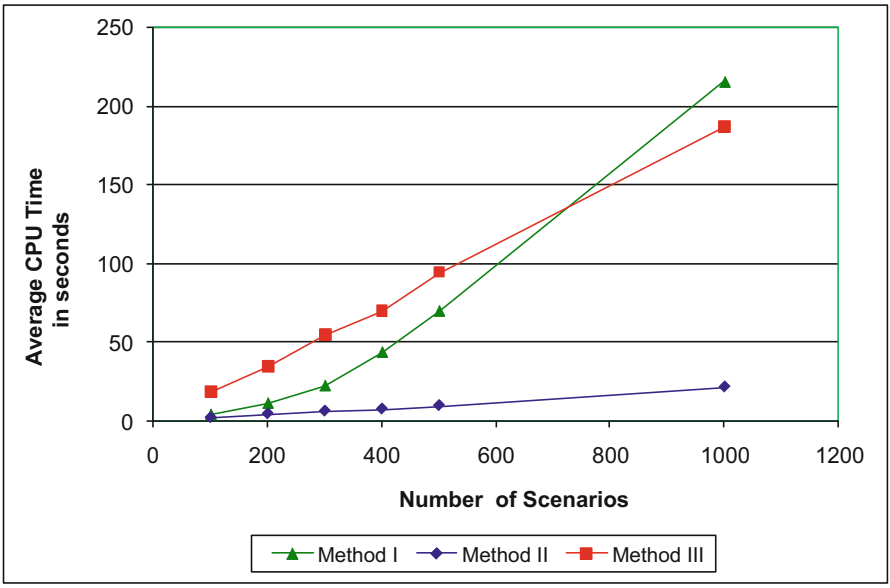


Fig. 3.3 Average CPU times required by the combination "L•H" × "(*ii*)" for 6-machine problems for various numbers of scenarios

increases, the CPU time required by Method II increases almost linearly, while that for Method I increases superlinearly. Although the CPU time for Method III also increases almost linearly, it does so at a much higher rate than that for Method II. Hence, the dominance of Method II becomes more prominent with an increase in the number of scenarios. Also, note that Method III begins to dominate Method I for larger number of scenarios.

Next, we investigated the impact of adding the RF and DP inequalities to Method II. There are several options that we can consider. The default option is Method II with the FBC inequalities. We can either add to this default option the RF and DP inequalities separately, or we can add them both at the same time. Another aspect to consider is the set of machines to which these inequalities are applied. We considered two options in this respect, namely, their application to only the HNV machines, or to only the LNV machines. Our preliminary investigation showed that the application of the RF and DP inequalities to the HNV machines results in the addition of a large number of extra variables and constraints to the Stage-I master problem, which leads to greater CPU times in comparison with the default option. Therefore, in the following discussion, we only consider the addition of the RF and DP inequalities to the LNV machines. To compare the performances of different model configurations, we fixed the number of scenarios at 400. The average CPU times and the ratio between the values of the LP solution and the 1. best-found solution, under different job-visiting patterns and cases of routing flexibility, are summarized in Tables 3.3 and 3.4.

Table 3.3 Average CPU times and percentage improvements over Model A

Model			A (default)	B (default + RF)	C (default + DP)	D (default + RF+ DP)
Number of machines	Job-visiting pattern	Routing flexibility	Average CPU time (in seconds), percentage improvement			
6	$L^+ \bullet H^-$	ρ_1	7.48	7.7	7.64	7.7
		ρ_2	17.48	22.12	25.52	23.6
		ρ_3	1.74	1.77	**1.74, 0.1 %**	1.77
	L•H	ρ_1	27.05	28.31	**26.4, 2.4 %**	28.32
		ρ_2	7.75	9.73	9.36	9.85
		ρ_3	3.09	3.09	3.11	3.13
	$L^- \bullet H^+$	ρ_1	130.12	**123.66, 5.0 %**	**124.15, 4.6 %**	**123.95, 4.7 %**
		ρ_2	4.16	4.49	4.25	4.49
		ρ_3	2.8	**2.77, 0.8 %**	2.81	**2.78, 0.8 %**
8	$L^+ \bullet H^-$	ρ_1	3.81	4.08	3.84	3.95
		ρ_2	5.78	8.97	6.85	9.44
		ρ_3	0.93	0.94	0.95	0.94
	L•H	ρ_1	29	**22.86, 21.2 %**	33.37	**22.94, 20.9 %**
		ρ_2	4.67	5.74	6.92	6.6
		ρ_3	0.97	**0.94, 3.0 %**	**0.97, 0.1 %**	**0.94, 3.1 %**
	$L^- \bullet H^+$	ρ_1	176.92	196.35	**160.16, 9.5 %**	195.94
		ρ_2	2.93	**2.81, 4.0 %**	**2.78, 5.1 %**	3.03
		ρ_3	0.76	0.77	0.78	0.77
10	$L^+ \bullet H^-$	ρ_1	178.09	**74.69, 58.1 %**	252.17	**76.01, 57.3 %**
		ρ_2	18.91	29.81	51.57	42.7
		ρ_3	1.21	1.23	1.29	1.23
	L•H	ρ_1	2227.97	**1066.2, 52.1 %**	**2170.97, 2.6 %**	**1086.73, 51.2 %**
		ρ_2	29.2	47.37	41.36	36.74
		ρ_3	1.6	1.62	1.85	1.62
	$L^- \bullet H^+$	ρ_1	549.05	**476.28, 13.3 %**	662.12	**536.86, 2.2 %**
		ρ_2	11.21	**10.89, 2.9 %**	15.28	15.37
		ρ_3	1.55	**1.51, 2.4 %**	**1.54, 1.1 %**	**1.51, 2.6 %**
Total average	127.64	79.88	134.06	83.29		

We highlight in bold, in Table 3.3, the CPU times that turn out to be shorter than that for the default model (A) for a given combination of job-visiting pattern and routing flexibility. For each such case, the percentage improvement in CPU time over the default model is also presented along with the CPU time. In view of these results, we can make the following observations: Simlarly a higher ratio between the values of the LP solution and the best-found solution obtained for a method over default is highlighted in Table 3.4

Table 3.4 Comparing quality of LP relaxation

Model			A (default)	B (default + RF)	C (default + DP)	D (default + RF+ DP)
Number of machines	Job-visiting pattern	Routing flexibility	LP relaxation value/Best solution found			
6	$L^+ \bullet H^-$	ρ_1	92.52 %	92.52 %	92.52 %	92.52 %
		ρ_2	93.52 %	**94.13 %**	**93.85 %**	**94.13 %**
		ρ_3	95.00 %	95.00 %	95.00 %	95.00 %
	L•H	ρ_1	94.00 %	94.00 %	94.00 %	94.00 %
		ρ_2	92.62 %	**92.96 %**	**92.80 %**	**92.96 %**
		ρ_3	91.58 %	91.58 %	91.58 %	91.58 %
	$L^- \bullet H^+$	ρ_1	94.45 %	94.45 %	94.45 %	94.45 %
		ρ_2	92.59 %	**92.78 %**	**92.75 %**	**92.78 %**
		ρ_3	92.46 %	92.46 %	92.46 %	92.46 %
8	$L^+ \bullet H^-$	ρ_1	93.92 %	93.92 %	93.92 %	93.92 %
		ρ_2	92.88 %	**93.25 %**	**93.10 %**	**93.25 %**
		ρ_3	93.95 %	93.95 %	93.95 %	93.95 %
	L•H	ρ_1	94.83 %	94.83 %	94.83 %	94.83 %
		ρ_2	92.74 %	**93.00 %**	**92.91 %**	**93.00 %**
		ρ_3	94.48 %	94.48 %	94.48 %	94.48 %
	$L^- \bullet H^+$	ρ_1	93.24 %	93.24 %	93.24 %	93.24 %
		ρ_2	93.70 %	**93.80 %**	**93.78 %**	**93.80 %**
		ρ_3	93.99 %	93.99 %	93.99 %	93.99 %
10	$L^+ \bullet H^-$	ρ_1	94.19 %	94.19 %	94.19 %	94.19 %
		ρ_2	92.98 %	**93.29 %**	**93.20 %**	**93.29 %**
		ρ_3	94.31 %	94.31 %	94.31 %	94.31 %
	L•H	ρ_1	93.59 %	93.59 %	93.59 %	93.59 %
		ρ_2	93.78 %	**94.03 %**	**93.94 %**	**94.03 %**
		ρ_3	93.12 %	93.12 %	93.12 %	93.12 %
	$L^- \bullet H^+$	ρ_1	92.38 %	92.38 %	92.38 %	92.38 %
		ρ_2	92.69 %	**92.87 %**	**92.82 %**	**92.87 %**
		ρ_3	94.47 %	94.47 %	94.47 %	94.47 %

1. On average, the addition of the RF inequalities alone (Model B) and the addition of both types of inequalities (Model D) help in achieving a shorter CPU time. This is particularly true when routing flexibility occurs on the HNV machines (Case ρ_1) because it leads to fewer conflicts for a large number of operations. This phenomenon appears to become more prominent with an increase in the number of machines, where savings of up to 58.1 % are achieved by Model B and up to 57.3 % by Model D for the 10-machine problems.
2. The benefit of adding the RF and DP inequalities is more evident for the harder problems, i.e., when the default model (Model A) takes a longer time to solve a problem, the addition of the RF and DP inequalities is more likely to help reduce the CPU time.

3.5 Heuristic Methods for the Solution of the NFJS Problem

Several heuristic methods can be used for the solution of the NFJSP that rely on work presented above. Six such viable procedures are described and investigated below.

1. Expected Value Problem Heuristic (EVP)

1. Assume processing times to be deterministic and equal to their expected values, and solve the model to optimality record the solution.
2. Evaluate solution using all scenarios.

2. Mixed Heuristic (Mixed)

Note that, even if all the scenarios are considered in the determination of budget over-runs, the resulting model is still relatively easy to solve (as the number of relevant constraints/variables is equal to *NS*). Hence, we can consider all scenarios in the determination of budget over-runs, while using expected processing times for the determination of job completion times. We call the resulting model a mixed-type model. Since this is a closer approximation of the original problem, we expect it to yield better solutions than the expected value heuristic.

For large-sized problem instances, even the mixed-type model becomes very difficult to solve. Therefore, we further relax the sequencing variables to be continuous and solve the mixed-type model to obtain assignment decisions. Then, we determine job sequences by considering the outcomes of all scenarios.

1. Solve the mixed-type model with sequencing variables relaxed as continuous; (assignment step).
2. Determine the processing sequences using the assignment decisions fixed in Step 1; (sequencing step).

3. Mixed + Shortest Processing Time (SPT)

The second step of the mixed heuristic is still difficult to solve for large-sized problems, hence we can apply the SPT dispatching rule to determine the job sequence. That is, whenever a machine is released by a previous operation, we choose the operation that has the shortest processing time on that machine from among the waiting operations. Note that, if an operation can be processed on multiple machines, it is considered to be waiting on all the compatible machines. Its assignment is determined based on which machine first chooses it as the next operation to be processed. Note that the SPT rule is based on the expected processing times.

4. Mixed + Least Work Remaining (LWKR)

This approach is similar to "Mixed + SPT," except that we use the least-work-remaining-first rule to dispatch operations.

5 and 6. Mixed + Shifting Bottleneck Heuristic (SBN)

As a first step, we use Step 1 of the Mixed Heuristic to determine the assignment of operations to the machines. The remaining sequencing problem is modeled as a disjunctive graph. In the beginning, all disjunctive arcs are relaxed and job completion times are recorded and regarded as due dates. The ready time and due date of each operation are determined by following the forward and backward passes along the critical paths (for the completion time problem, there are usually multiple critical paths). Next, each machine is considered individually to fix its operation sequence using a heuristic rule. The machine that yields the largest change in the total completion time is chosen, and the corresponding sequence (set of disjunctive arcs) is fixed. The procedure continues until all the machines are sequenced. Note that after the sequence of operations on each machine is fixed, a re-sequencing step is implemented by adjusting the sequences of operations on previously fixed machines.

We employ the following two heuristic rules to determine the sequence in which to process the jobs on a machine:

5. SBN_ATC Determine the following Apparent Tardiness Cost (ATC) priority index (Pinedo and Singer 1999):

$$I_{ij} = \sum_{k=1}^{N} \frac{1}{p_{ij}} \exp\left(-\frac{d_{ij}^k - p_{ij} + \left(r_{ij} - t\right)^+}{K\overline{p}}\right), \quad \forall i \in M, j = 1, \ldots, N,$$

where t is the scheduling time, $K = 2$, and $\overline{p}$ is the average of the processing times of the jobs assigned to machine i, and ready time r_{ij} and local due date d_{ij}^k of operation j of job k assigned to machine i are determined as explained in Pinedo and Singer (1999).

6. SBN_PRTT Choose the operation that has the lowest value of Chu and Portmann (1992):

$$PRTT_{ij} = \max\left(r_{ij},\ t\right) + \max\left\{\max\left\{r_{ij},\ t\right\} + p_{ij},\quad \min_k\left\{d_{ij}^k\right\}\right\},$$

$$\forall i \in M, j = 1, \ldots, N.$$

This is a single machine sequencing rule, and we use the values of r_{ij} and d_{ij}^k as determined above. Additionally, insert the earliest available operation, if the operation chosen by the above rule leaves enough idle time before itself.

The relative performances of these heuristic methods are presented in Tables 3.5 and 3.6 for 6 machines (with three jobs) and 10 machines (with 15 jobs), respectively. Optimal gap is determined with respect to the solution of mixed-method, which serves as a lower bound. Note that, although EVP gives the best results with the least CPU time for the first set of problems, it becomes impractical for larger-sized problems (second set) due to its large number of binary sequencing variables. Mixed + LWKR heuristic gives the best results on larger-sized problem instances.

Table 3.5 Results on 6-machine problems (with 3 jobs)

Approach	Optimality gap	CPU time (seconds)
EVP	0.49 %	0.33
Mixed	–	0.27 (Step 1)
Mixed + SPT	1.46 %	1.63
Mixed + LWKR	1.42 %	1.64
Mixed + SBN_ATC	1.89 %	0.91
Mixed + SBN_PRTT	**1.27 %**	0.90

Table 3.6 Results on 10-machine problems (with 15 jobs)

Approach	Objective value	CPU time (seconds)
Mixed	2049.72	2.15 (Step 1)
Mixed + LWKR	**2796.85 (37.16 %)**	13.12
Mixed + SBN_ATC	2872.97 (40.79 %)	45.35
Mixed + SBN_PRTT	2804.47 (37.47 %)	47.32

3.6 Concluding Remarks

In this chapter, we have presented a stochastic programming approach for the NFJS (Network of Flexible Job Shops) problem. This problem arises in a distributed fabrication environment that has recently emerged to serve the evolving needs of the high investment, low volume MEMS industry. The problem is modeled as a two-stage stochastic program with recourse, where the uncertainty in processing times is captured using scenarios. The first-stage routing and sequencing variables are binary whereas the second-stage completion time and budget over-run variables are continuous. Since the NFJS problem lacks relatively complete recourse, the first-stage solution can be infeasible to the second-stage problem in that it might generate subtours, violate the re-entrant flow conditions, or create a deadlock. In the standard L-shaped method, feasibility cuts are iteratively added to the first-stage problem upon the discovery of these infeasibilities. As an alternative, we have provided certain expected-value-scenario-based inequalities to induce feasibility of the second-stage problem that greatly help reduce the effort required by the L-shaped method. To further tighten the first-stage problem formulation, we have also developed three types of valid inequalities: flow-balance constraints, re-entrant flow-based constraints, and deadlock prevention constraints. Our computational results reveal that: (a) our decomposition approach is substantially superior to the direct solution of the NFJSP using CPLEX; (b) the expected-value-scenario-based inequalities are significantly more effective than the use of standard feasibility cuts in the master problem; and (c) the judicious additional use of the re-entrant flow and deadlock prevention inequalities in conjunction with the expected-value-scenario-based inequalities further improves the overall algorithmic performance, particularly for more difficult problem instances. Furthermore, we have proposed heuristic methods for the solution relatively larger instances of NFJS and have presented results of their implementation.

Acknowledgements This work has been supported by the National Science Foundation under Grant CMMI-0856270.

References

Bagga PC (1970) N jobs, 2 machines sequencing problems with stochastic service times. Oper Res 7:184–197

Benders JF (1962) Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik 4:238–252

Birge JR, Louveaux F (2000) Introduction to stochastic programming. Springer, New York

Brandimarte P (1993) Routing and scheduling in a flexible job shop by tabu search. Ann Oper Res 41:157–183

Chu C, Portmann MP (1992) Some new efficient method to solve the $n \mid T \mid r_i \mid \Sigma_i w_i T_i$ problem. Eur J Oper Res 58:404–413

Cunningham AA, Dutta SK (1973) Scheduling jobs with exponentially distributed processing times on two machines of a flow shop. Nav Res Log Q 16:69–81

Dauzère-Pérès S, Paulli J (1997) An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Ann Oper Res 70:281–306

Elmaghraby SE, Thoney KA (1999) The two machine stochastic flowshop problem with arbitrary processing time distributions. IIE Trans 31:467–477

Foley RD, Suresh S (1984) Stochastically minimizing the makespan in flow shops. Nav Res Log Q 31:551–557

Forst FG (1983) Minimizing total expected costs in the two machine, stochastic flow shop. Oper Res Lett 2:58–61

Gascon A, Lefrancois P, Cloutier L (1998) Computer-assisted multi-item, multi-machine and multi-site scheduling in a hardwood flooring factory. Comput Ind 36:231–244

Gnoni MG, Iavagnilio R, Mossa G, Mummolo G, Di Leva A (2003) Production planning of multi-site manufacturing system by hybrid modelling: a case study from the automotive industry. Int J Prod Econ 85:251–262

Golenko-Ginzburg D, Gonik A (1997) Using "look-ahead" techniques in job-shop scheduling with random operations. Int J Prod Econ 50:13–22

Golenko-Ginzburg D, Gonik A (2002) Optimal job-shop scheduling with random operations and cost objectives. Int J Prod Econ 76:147–157

Golenko-Ginzburg D, Kesler S, Landsman Z (1995) Industrial job-shop scheduling with random operations and different priorities. Int J Prod Econ 40:185–195

Guinet A (2001) Multi-site planning: a transshipment problem. Int J Prod Econ 74:21–32

Hutchison J, Leong K, Snyder D, Ward P (1991) Scheduling approaches for random job shop flexible manufacturing systems. Int J Prod Res 29(11):1053–1067

Iwata K, Murotsu Y, Oba F, Okamura K (1980) Solution of large-scale scheduling problems for job-shop type machining systems with alternative machine tools. CIRP Ann Manuf Technol 29:335–338

Jia C (1998) Minimizing variation in a stochastic flowshop. Oper Res Lett 23:109–111

Jia HZ, Nee AYC, Fuh JYH, Zhang YF (2003) A modified genetic algorithm for distributed scheduling problems. J Intell Manuf 14(3–4):351–362

Kamburowski J (1999) Stochastically minimizing the makespan in two-machine flowshops without blocking. Eur J Oper Res 112:304–309

Kamburowski J (2000) On three machine flowshops with random job processing times. Eur J Oper Res 125:440–449

Kim Y-D (1990) A comparison of dispatching rules for job shops with multiple identical jobs and alternative routings. Int J Prod Res 28(5):953–962

Ku PS, Niu SC (1986) On Johnson's two-machine flowshop with random processing times. Oper Res 34:130–136

Kutanoglu E, Sabuncuoglu I (2001) Experimental investigation of iterative simulation-based scheduling in a dynamic and stochastic job shop. J Manuf Syst 20(4):264–279

Lai T-C, Sotskov YN, Sotskova N, Werner F (2004) Mean flow time minimization with given bounds of processing times. Eur J Oper Res 159:558–573

Lee C-Y, Chen Z-L (2001) Machine scheduling with transportation considerations. J Sched 4:3–24

Lin JT, Chen Y-Y (2006) A multi-site supply network planning problem considering variable time buckets–A TFT-LCD industry case. Int J Adv Manuf Technol 33:1031–1044

Luh PB, Chen D, Thakur LS (1999) An effective approach for job-shop scheduling with uncertain processing requirements. IEEE Trans Robot Autom 15(2):328–339

Makino T (1965) On a scheduling problem. J Oper Res Soc Jpn 8:32–44

Mastrolilli M, Gambardella LM (2000) Effective neighbourhood functions for the flexible job shop problem. J Sched 3:3–20

Miller C, Tucker A, Zemlin R (1960) Integer programming formulation of traveling salesman problems. J ACM 7:326–329

Mittal BS, Bagga PC (1977) A priority problem in sequencing with stochastic service times. Oper Res 14:19–28

Nasr N, Elsayed EA (1990) Job shop scheduling with alternative machines. Int J Prod Res 28(9):1595–1609

Pezzella F, Morganti G, Ciaschetti G (2008) A genetic algorithm for the flexible job-shop scheduling problem. Comput Oper Res 35:3202–3212

Pinedo M (1982) Minimizing the expected makespan in stochastic flow shops. Oper Res 30: 148–162

Pinedo M, Singer M (1999) A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. Nav Res Log 48:1–17

Prasad VR (1981) $n \times 2$ flowshop sequencing problem with random processing times. Oper Res 18:1–14

Roux W, Dauzère-Pérès S, Lasserre JB (1999) Planning and scheduling in a multi-site environment. Prod Plan Control 10(1):19–28

Sarin SC, Sherali HD, Bhootra A (2005) New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints. Oper Res Lett 33(1):62–70

Sauer J, Freese T, Teschke T (2000) Towards agent based multi-site scheduling. In: Proceedings of the ECAI 2000 workshop on new results in planning, scheduling, and design, pp 123–130

Sherali HD, Sarin SC, Tsai P (2006) A class of lifted path and flow-based formulations for the asymmetric traveling salesman problem with and without precedence constraints. Discret Optim 3(1):20–32

Singer M (2000) Forecasting policies for scheduling a stochastic due date job shop. Int J Prod Res 38(15):3623–3637

Subramaniam V, Lee GK, Ramesh T, Hong GS, Wong YS (2000) Machine selection rules in a dynamic job shop. Int J Adv Manuf Technol 16:902–908

Talwar TT (1967) A note on sequencing problems with uncertain job times. J Oper Res Soc Jpn 9:93–97

Tavakkoli-Moghaddam R, Jolai F, Vaziri F, Ahmed PK, Azaron A (2005) A hybrid method for solving stochastic job shop scheduling problems. Appl Math Comput 170:185–206

Timpe CH, Kallrath J (2000) Optimal planning in large multi-site production networks. Eur J Oper Res 126(2):422–435

van Slyke R, Wets RJ-B (1969) L-shaped linear programs with applications to optimal control and stochastic programming. SIAM J Appl Math 17:638–663

Varadarajan A (2006) Stochastic scheduling for a network of MEMS job shops. Ph.D. Dissertation, Grado Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, Virginia

Wang L, Zhang L, Zheng D-Z (2005) A class of hypothesis-test based genetic algorithms for flowshop scheduling with stochastic processing times. Int J Adv Technol 25(11–12):1157–1163

Weiss G (1982) Multiserver stochastic scheduling. In: Dempster M, Lenstra JK, Rinooy-Kan A (eds) Deterministic and stochastic scheduling. D. Reidel, Dordrecht, Holland

Xia W, Wu Z (2005) An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. Comput Ind Eng 48:409–425

Yoshitomi Y (2002) A genetic algorithm approach to solving stochastic job-shop scheduling problems. Int Trans Oper Res 9:479–495

Chapter 4
A Free-Slack-Based Genetic Algorithm for the Robotic Cell Problem with Controllable Processing Times

Mohammed Al-Salem, Mohamed Haouari, Mohamed Kharbeche, and Wael Khallouli

Abstract We present a novel genetic algorithm for the Robotic Cell Problem with controllable processing times. This challenging problem arises in an automated production cell that consists of m consecutive machines as well as a material handling robot. The problem requires finding the operations processing times, job assignment, and robot movements. The objective is to minimize the makespan subject to a budget constraint. We describe a free-slack-based genetic algorithm for the linear resource consumption case. We present the results of a computational study and we provide evidence that the proposed algorithm consistently outperforms MIP-based heuristics from the literature.

Keywords Flexible manufacturing • Robotic cell scheduling • Controllable processing • Genetic algorithms • Makespan

M. Al-Salem (✉)
Department of Mechanical and Industrial Engineering, College of Engineering, Qatar University, Doha, Qatar
e-mail: alsalem@qu.edu.qa

M. Haouari
Department of Mechanical and Industrial Engineering, College of Engineering, Qatar University, Doha, Qatar
e-mail: mohamed.haouari@qu.edu.qa

M. Kharbeche
Qatar Road Safety Studies Center, Qatar University, Doha, Qatar
e-mail: mkharbec@qu.edu.qa

W. Khallouli
Department of Mechanical and Industrial Engineering, College of Engineering, Qatar University, Doha, Qatar
e-mail: w.khallouli@qu.edu.qa

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_4

4.1 Introduction

With the increasing attention to flexible manufacturing systems (FMSs), there is a growing interest from researchers to develop heuristic and optimization approaches for scheduling these complex systems. A clear evidence from the FMS literature is that the processing times are often considered to be constant parameters, and part of the problem input. However, in many real-life situations, the processing times may be controllable (that is, increased or decreased) by allocating resources (energy, workforce, money). In such situations, identifying the operational tradeoffs between the processing times and the resource allocation is of crucial importance. Indeed, an increase in the operations processing times will reduce the tools wear while improving the cost. On the other hand, decreasing the operations processing times will cause excessive tool wear with additional energy which will translate into higher costs. Therefore, since the processing times and the resource allocation constitute important factors in FMS, they should be considered as *decision variables* and therefore should be determined as part of the optimization process.

In this chapter, we propose a genetic algorithm for the Robotic Cell Problem (RCP) with controllable processing times. We provide empirical evidence that the new proposed genetic algorithm consistently produces better solutions than MIP-based heuristics from the literature.

The remainder of this chapter is organized as follows. In Sect. 4.2, we provide a formal description of the problem as well as a valid mathematical formulation. Also, we briefly overview the literature on both robotic cells and controllable processing times scheduling problems. In Sect. 4.3, we present a free-slack-based genetic algorithm for the case of linear resource consumption function. In Sect. 4.4, we present the results of a computational study that was conducted to assess the performance of the proposed approach. Finally, Sect. 4.5 provides a summary of the chapter along with some concluding remarks and directions for future research.

4.2 Problem Description

The RCP with controllable processing times (RCPCPT) can be formally described as follows. We are given a set J of n jobs where each job has to be processed in a robotic cell. The robotic cell consists of a set of m machines M_1, M_2,..., M_m, an input buffer M_0, an output buffer M_{m+1}, and a handling robot that is used for transferring the jobs between machines. At time $t = 0$, all jobs are available at the input device M_0. Each job $j \in J$ has to be processed non preemptively on machines M_1, M_2,..., M_m, in that order, and then transferred to the output device M_{m+1}. The robot can transfer at most one job at any time and the duration of a robot move from M_i to M_h $(i, h = 0, \ldots, m + 1)$ is deterministic and requires τ_{ih} units of time. The machines have neither input nor output buffering facilities. Consequently, after processing a job j on machine M_i $(i = 1, \ldots, m)$, the latter remains blocked until the robot picks up j and transfers it to the subsequent machine M_{i+1}. Such a move could

only be performed if machine M_{i+1} is free (that is, no job is being processed by or waiting at M_{i+1}). It is noteworthy that, because of the blocking constraint, passing is not possible and therefore only permutation schedules are considered. Furthermore, at any time each machine can process at most one job and each job can be processed on at most one machine.

An important feature of the investigated problem is that the processing time p_{ij} of operation O_{ij} of job $j \in J$ on machine M_i ($i = 1, \ldots, m$) is not a priori fixed. For each operation, the processing time p_{ij} varies between a non-compressed processing time $p_{ij}^{\max}$ and a compressed (minimum) processing time $p_{ij}^{\min}$:

$$p_{ij}^{\min} \leq p_{ij} \leq p_{ij}^{\max}, \qquad \forall j \in J \text{ and } i = 1, \ldots, m. \tag{4.1}$$

For the linear resource consumption function case, the processing time p_{ij} is assumed to be a decreasing function of the *acceleration cost* c_{ij} that is allocated to the processing of O_{ij}. More precisely, the data of each operation O_{ij} includes three parameters: a non-compressed (maximum) processing time $p_{ij}^{\max}$, a compressed (minimum) processing time $p_{ij}^{\min}$, and a compression rate a_{ij}. Then, the processing times of the operations are given by:

$$p_{ij} = p_{ij}^{\max} - a_{ij}c_{ij}, \ \forall j \in J \text{ and } i = 1, \ldots, m. \tag{4.2}$$

where the acceleration costs satisfy:

$$0 \leq c_{ij} \leq c_{ij}^{\max} \equiv (p_{ij}^{\max} - p_{ij}^{\min})/a_{ij}, \qquad \forall j \in J \text{ and } i = 1, \ldots, m. \tag{4.3}$$

It is noteworthy this linear resource consumption function is very popular in the scheduling literature dealing with controllable processing times [see Biskup and Cheng (1999), Janiak (1991), Wang and Xia (2007) and Zdrzałka (1991), to quote just a few].

4.2.1 Mathematical Formulation

In this section, we first briefly describe the decision variables, followed by a nonlinear mixed-integer programming formulation proposed by Al-Salem et al. (2015).

Prior to describing the decision variables, we introduce an additional notation. We define the set Π, as the set of the robot loaded moves. A robot operation that corresponds to a transfer of the kth job from M_i to M_{i+1} is denoted by θ_{ik} ($i = 0, \ldots, m$, $k = 1, \ldots, n$). Clearly, robot moves operations are interrelated by precedence relationships. More precisely, because of the blocking constraints, θ_{ik} should be preceded by operation $\theta_{i+1,k-1}$ ($i = 0, \ldots, m$, $k = 2, \ldots, n$). Also, as a consequence of the flow shop constraint, θ_{ik} should be preceded by operation $\theta_{i-1,k}$ ($i = 1, \ldots, m, k = 1, \ldots, n$). For each $\theta_{ik} \in \Pi$, we denote by pred(θ_{ik}) and succ(θ_{ik}) the sets of predecessors and successors of θ_{ik}, respectively.

Decision variables:

x_{kj} : binary variable that takes value 1 if job j is assigned to the kth position in the schedule, and 0 otherwise, $\forall k = 1, \ldots, n, j \in J$
y_{ik}^{lh} : binary variable that takes value 1 if the robot loaded move θ_{ik} is achieved before the robot move θ_{lh}, and 0 otherwise, $\forall \theta_{ik}, \theta_{lh} \in \Pi$
t_{ik} : starting time of operation θ_{ik}, $\forall i = 0, \ldots, m, k = 1, \ldots, n$
p_{ij} : processing time of job j on machine M_i, $\forall i = 1, \ldots, m, j \in J$.

Model formulation:
Using these definitions, the RCPCPT is formulated as the following:

$$\textbf{MINLP:} \quad \text{Minimize } t_{mn} + \tau_{m,m+1} \tag{4.4}$$

subject to:

$$\sum_{k=1}^{n} x_{kj} = 1, \ \forall j \in J, \tag{4.5}$$

$$\sum_{j=1}^{n} x_{kj} = 1, \ \forall j \in J, \tag{4.6}$$

$$t_{i+1,k} \geq t_{ik} + \tau_{i,i+1} + \sum_{j=1}^{n} p_{i+1,j} x_{kj},$$

$$\forall \quad k = 1, \ldots, n, \ i = 0, \ldots, m-1, \tag{4.7}$$

$$t_{i-1,k} \geq t_{i,k-1} + \tau_{i,i+1} + \tau_{i+1,i-1},$$

$$\forall \quad k = 2, \ldots, n, \ i = 1, \ldots, m, \tag{4.8}$$

$$\sum_{\theta_{lh} \in \text{pred}(\theta_{ik})} y_{lh}^{ik} = 1, \ \forall \theta_{ik} \in \Pi \setminus \{\theta_{01}\}, \tag{4.9}$$

$$\sum_{\theta_{lh} \in \text{succ}(\theta_{ik})} y_{ik}^{lh} = 1, \ \forall \theta_{ik} \in \Pi \setminus \{\theta_{mn}\}, \tag{4.10}$$

$$t_{i-1,k} \geq t_{i+1,k-1} + \tau_{i+1,i+2} + \tau_{i+2,i-1}$$
$$+ M(y_{i+1,k-1}^{i-1,k} - 1), \ \forall k = 2, \ldots, n, \ i = 1, \ldots, m-1, \tag{4.11}$$

$$t_{i+1,k-1} \geq t_{i-1,k} + \tau_{i-1,i} + \tau_{i,i+1}$$
$$+ M(y_{i-1,k}^{i+1,k-1} - 1), \ \forall k = 2, \ldots, n, \ i = 1, \ldots, m-1, \tag{4.12}$$

$$t_{0,k} \geq t_{m,k-p} + \tau_{m,m+1} + \tau_{m+1,0}$$
$$+ M(y_{m,k-p}^{0,k} - 1), \ \forall k = 2, \ldots, n, \ p = 1, \ldots, m-1, \tag{4.13}$$

$$t_{0,k+p} \geq t_{m,k-1} + \tau_{m,m+1} + \tau_{m+1,0}$$
$$+ M(y_{m,k-1}^{0,k+p} - 1), \ \forall k = 2, \ldots, n, \ p = 1, \ldots, m-1, \tag{4.14}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\frac{p_{ij}}{a_{ij}} \geq \sum_{i=1}^{m}\sum_{j=1}^{n}\frac{p_{ij}^{\max}}{a_{ij}} - B, \tag{4.15}$$

$$p_{ij}^{\min} \leq p_{ij} \leq p_{ij}^{\max},\ \forall i = 1,\ldots,m,\ j \in J, \tag{4.16}$$

$$t_{ik} \geq 0,\ \forall i = 0,\ldots,m,\ k = 1,\ldots,n, \tag{4.17}$$

$$x_{kj} \in \{0,1\},\ \forall k = 1,\ldots,n,\ j \in J, \tag{4.18}$$

$$y_{ik}^{lh} \in \{0,1\},\ \forall \theta_{ik}, \theta_{lh} \in \Pi. \tag{4.19}$$

The objective (4.4) is to minimize the makespan. Constraints (4.5) and (4.6) require that each job is assigned to exactly one position in the schedule, and each position is assigned to exactly one job, respectively. Constraint (4.7) is the flow shop constraint: it requires that $\theta_{i+1,k}$ is scheduled after achieving θ_{ik} and processing operation O_{ik} on M_i. Constraint (4.8) is the blocking constraint: $\theta_{i-1,k}$ is scheduled after $\theta_{i,k-1}$. Constraint (4.9) enforces that each robot operation (but, θ_{01}) has exactly one successor. Similarly, Constraint (4.10) requires that each robot operation (but, θ_{mn}) has exactly one predecessor. Constraints (4.11)–(4.14) ensure that the precedence constraints between particular robot operations must be satisfied. Constraint (4.15) requires that the total acceleration costs should not exceed the preset budget and (4.16) sets the upper and lower bounds for the processing times. Finally, (4.17)–(4.19) imposes that the time variables are continuous and the x- and y-variables are binary, respectively.

For more details about the mixed-integer nonlinear program (MINLP) and its linearization, we refer the reader to Al-Salem et al. (2015).

4.2.2 Literature Review

The RCPCPT arises in FMSs, which are highly automated production systems capable of producing a wide variety of job types. In fact, one of the most crucial operational problems in FMSs is the development of effective schedules considering jobs, machines, and transportation devices in order to provide a proper coordination of the production sequencing and time allocation of all required resources. During the last decades, the rapid development of robotic cells in various manufacturing industrial setting has prompted the investigation of an ever growing number of new scheduling problems. We refer to the comprehensive book of Dawande et al. (2007) for a review of sequencing and scheduling problems arising in robotic cells.

However, at this point it is worth emphasizing that the great majority of previously investigated robotic cell scheduling problems deal with *cyclic scheduling problems* with constant processing times and machines producing a family of similar parts, in a steady-state. Nevertheless, few recent papers addressed a non-cyclic multiple-part-type RCP (with constant processing times). Indeed, Carlier et al. (2010) proposed an approximate decomposition algorithm to the RCP. The proposed approach decomposes the problem into two scheduling problems: a flow shop

problem with blocking and transportation times and a single-machine problem (that corresponds to the robot sequencing) with precedence constraints, time lags, and setup times. Each of these two problems is solved using an exact branch-and-bound algorithm. Furthermore, Kharbeche et al. (2011) proposed an exact branch-and-bound algorithm approach for the same problem and found that instances with up to 16 jobs and 5 machines can be optimally solved.

To the best of our knowledge, and despite its practical relevance to FMSs, the literature on the RCPCPT is void. However, we observe that during the last decade the topic of scheduling with controllable processing times has been drawing the attention of an ever growing number of researchers. In particular, most of the papers so far published deal with one-machine scheduling problems, and to a lesser extent with parallel machines, flow-shop, and job-shop problem. We refer to Shabtay and Steiner (2007) for a comprehensive review of these scheduling models. In addition to the numerous references quoted in this review paper, some further contributions in this area have been recently published. We quote Koulamas et al. (2010) who presented a unified analysis for a very general single-machine scheduling problem with controllable processing times. Also, Xu et al. (2010) a single-machine scheduling problem with release dates, due dates are arbitrary, and where the processing times can be controlled by allocating a common resource to each job through a nonlinear convex resource consumption function. The objective is to obtain a schedule and resource allocation, such that no job is tardy and the total resource consumption is minimized. The authors solved this problem using a tabu search algorithm. Finally, Akturk and Ilhan (2011) addressed a CNC machine scheduling with controllable processing times. In this case, processing times of the jobs on a CNC machine are controlled via machining conditions such that they can be increased or decreased at the expense of tooling cost. The authors addressed the problem of scheduling a set of jobs on a single CNC machine to minimize the sum of total weighted tardiness, tooling, and machining costs. They formulated the problem as nonlinear mixed-integer program and solved it using a heuristic approach.

In the sequel, a free-slack-based genetic algorithm is proposed for the problem under consideration. Then, a computational study is carried out to compare the performance of the new procedure against the MIP-based genetic algorithm by Al-Salem et al. (2015). This latter encompasses several novel features including, an original solution encoding as well as a mutation operator that requires iteratively solving mixed-integer formulations. We refer reader to the recent technical report by Al-Salem et al. (2015).

4.3 Free-Slack-Based Genetic Algorithm

In this section, we propose a free-slack-based genetic algorithm. The general framework of the algorithm includes four main steps. First, we start with an initial generation of processing times then, given the fixed processing times we find the sequence of jobs for the RCP yielding a near-optimal makespan. Given this

sequence, we improve its makespan by elongating the processing times of non-critical operations and accelerating the critical ones. Finally, the optimal robot moves are determined for the problem with the fixed sequence and the new updated processing times. An overview of the main steps is as follows:

- **Step 1:** Generation of initial processing times
- **Step 2:** Approximate solution of the Robotic Cell Problem RCP (with fixed processing times) using a genetic algorithm to compute a near-optimal sequencing of the machines and the robot.
- **Step 3:** Update of the processing times based on the free-slack procedure
- **Step 4:** Find the optimal robot moves

In the sequel, we provide detailed description of each step of this algorithm.

4.3.1 Initial Processing Times Generation

In the first step, we generate an initial $m \times n$ matrix of processing times $P = (p_{ij})$ with three different generation approaches: a deterministic method, a completely randomized method and a mixed-method (hereafter, referred by one by one method). All these generation methods are based on the compression rates a_{ij}. The aim of these generations is to diversify the initial solution, and meanwhile, begin with a good one.

Next, we describe the initial generation algorithms.

4.3.1.1 Deterministic Generation Method

A key element of this first approach is to sort the operations O_{ij} by the non-decreasing order of a_{ij} . We set the maximum (non-compressed) processing times p_{ij} to $p_{ij}^{\max}$. Starting from the operation with the highest rate to the operation with the lowest one, we increase each cost c_{ij} by one unit.

As given by Eq. (4.2), the corresponding processing time decreases by a_{ij}. The procedure is iterated until the preset budget B is fully consumed. The procedure is detailed in Algorithm 1.

4.3.1.2 Random Generation

This approach follows the same logic as the deterministic procedure. First, we start by sorting the list of operations according to the non-increasing order of a_{ij}. Then, we randomly select q operations from the list and then we choose the operation with the highest a_{ij} among them to update. The procedure is illustrated by Algorithm 2.

Algorithm 1 Generation of initial processing times: deterministic procedure

$S \leftarrow$ (List of all operations O_{ij})
Sort the list S according to non-increasing order of a_{ij}
Initialization
Set $p_{ij} \leftarrow p_{ij}^{max}$
Set $\bar{B} \leftarrow B - \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}$ ($\bar{B}$: remaining budget)
do
  for all elements of the list S **do**
    Select an operation O_{ij}
    if $p_{ij} \geq p_{ij}^{min} + a_{ij}$ **then**
      $p_{ij} \leftarrow p_{ij} - a_{ij}$
      $\bar{B} \leftarrow \bar{B} - 1$
    end if
  end for
while $\bar{B} \neq 0$

Algorithm 2 Generation of initial processing times: stochastic procedure

$S \leftarrow O$ (List of all operations O_{ij})
Sort the list S according to non-increasing order of a_{ij}
Initialization
Set $p_{ij} \leftarrow p_{ij}^{max}$
Set $\bar{B} \leftarrow B - \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}$ ($\bar{B}$: remaining budget)
do
  Randomly select q operations from S (if $|S| < q$ then select all operations in S)
  Select (among the q operations) O_{ij} that has the largest a_{ij}
  if $p_{ij} \geq p_{ij}^{min} + a_{ij}$ **then**
    $p_{ij} \leftarrow p_{ij} - a_{ij}$
    $\bar{B} \leftarrow \bar{B} - 1$
  end if
  if $p_{ij} = p_{ij}^{min}$ **then**
    $S \leftarrow S \setminus \{O_{ij}\}$
  end if
while $\bar{B} \neq 0$ or $S \neq \emptyset$

4.3.1.3 One by One Generation Method

This approach combines both the stochastic and the deterministic approaches. First, we sort the operations according to their compression rates a_{ij}. Then, at each time, we update the processing time of one operation selected from the sorted list and another one randomly selected until the preset budget is fully used. Algorithm 3 gives more details about this generation method.

Algorithm 3 Generation of initial processing times: one by one procedure

$S \leftarrow O$ (List of all operations O_{ij})
Sort the list S according to non-increasing order of a_{ij}
Initialization
Set$p_{ij} \leftarrow p_{ij}^{max}$
Set $\bar{B} \leftarrow B - \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}$ ($\bar{B}$: remaining budget)
do
  for all elements of the list **do**
    select an operation O_{ij}
    if $p_{ij} \geq p_{ij}^{min} + a_{ij}$ **then**
      $p_{ij} \leftarrow p_{ij} - a_{ij}$
      $\bar{B} \leftarrow \bar{B} - 1$
    end if
    Randomly draw a position i
    Select the operation O_{ij} at position i in the list
    if $p_{ij} \geq p_{ij}^{min} + a_{ij}$ **then**
      $p_{ij} \leftarrow p_{ij} - a_{ij}$
      $\bar{B} \leftarrow \bar{B} - 1$
    end if
  end for
while $\bar{B} \neq 0$

4.3.2 *Approximate Solution of the RCP (with Fixed Processing Times)*

Given the initial processing times obtained from the generation methods, we use a genetic algorithm of Carlier et al. (2010) to approximately find a good sequence of jobs. Let $C_{\max}^{\text{new}}$ be the approximate makespan obtained by the genetic algorithm. If $C_{\max}^{\text{new}}$ is better than the best makespan $C_{\max}^{\text{best}}$, we improve the solution under this sequence through the free slacks identification process. If not, we go to the next iteration to generate another chromosome.

4.3.3 *Update of the Processing Times*

In this step, we identify the set C of critical operations and the set F of non-critical operations having free slacks strictly larger than 1. Clearly, if the processing time of a non-critical operation $O_{ij} \in F$ is elongated by one unit of time then the makespan remains unchanged while the acceleration cost is reduced by a_{ij}. Hence, if we elongate the processing times of operations belonging to F, we reduce the total acceleration cost and, therefore, we save the unused budget. Furthermore, since the critical operations have an impact on the makespan $C_{\max}$, then it is possible to reduce the makespan by accelerating some critical operations using the collected budget.

We define δ_{ij} as the free-slack that corresponds to all operation O_{ij}. If $\delta_{ij} = 0$, the operation O_{ij} is a critical operation.

Algorithm 4 Update of the processing times of critical operations

$S \leftarrow C$ (List of all critical operations)
Sort the list S according to non-increasing order of a_{ij}
$B \leftarrow \tilde{B}$
do
  for all elements of the list S **do**
    if $p_{ij} \geq p_{ij}^{min} + a_{ij}$ **then**
      $p_{ij} \leftarrow p_{ij} - a_{ij}$
      $\tilde{B} \leftarrow \tilde{B} - 1$
    end if
  end for
while $\tilde{B} \neq 0$

To update the processing time, the following steps are used:

4.3.3.1 Elongate the Processing Times

For each operation O_{ij} belonging to F, we elongate the processing time using:

- A deterministic method: set $p_{ij} \leftarrow a_{ij}/2$
- A randomly elongation method: set $p_{ij} \leftarrow p_{ij} + \text{rand}(1, \delta_{ij} - 1)$

Let $\tilde{B}$ be the collected budget. This budget is equal to the difference between the preset budget B and the total new costs of O_{ij} after elongation.

4.3.3.2 Update of the Critical Operations

In this step, the collected budget is divided over the critical operations where these operations are sorted according to the non-increasing order of a_{ij}. In fact, the jobs with highest rates would have the priority to be accelerated. The critical tasks are updated using the Algorithm 4 described below.

4.3.4 Find the Optimal Robot Moves

Given the fixed sequence of jobs and the new processing times, we solve the robot moves scheduling problem. First, we approximately compute the robot moves using a list algorithm that is described in Kharbeche et al. (2011). If the makespan $C_{\max}^{R}$ is not improved then we invoke the exact optimization model (Kharbeche et al. 2011). In case of makespan improvement, we identify again the new critical and non-critical operations and we solve the robot moves problem.

4.3.5 Gant Chart of the Algorithm

The algorithm is illustrated in Fig. 4.1.

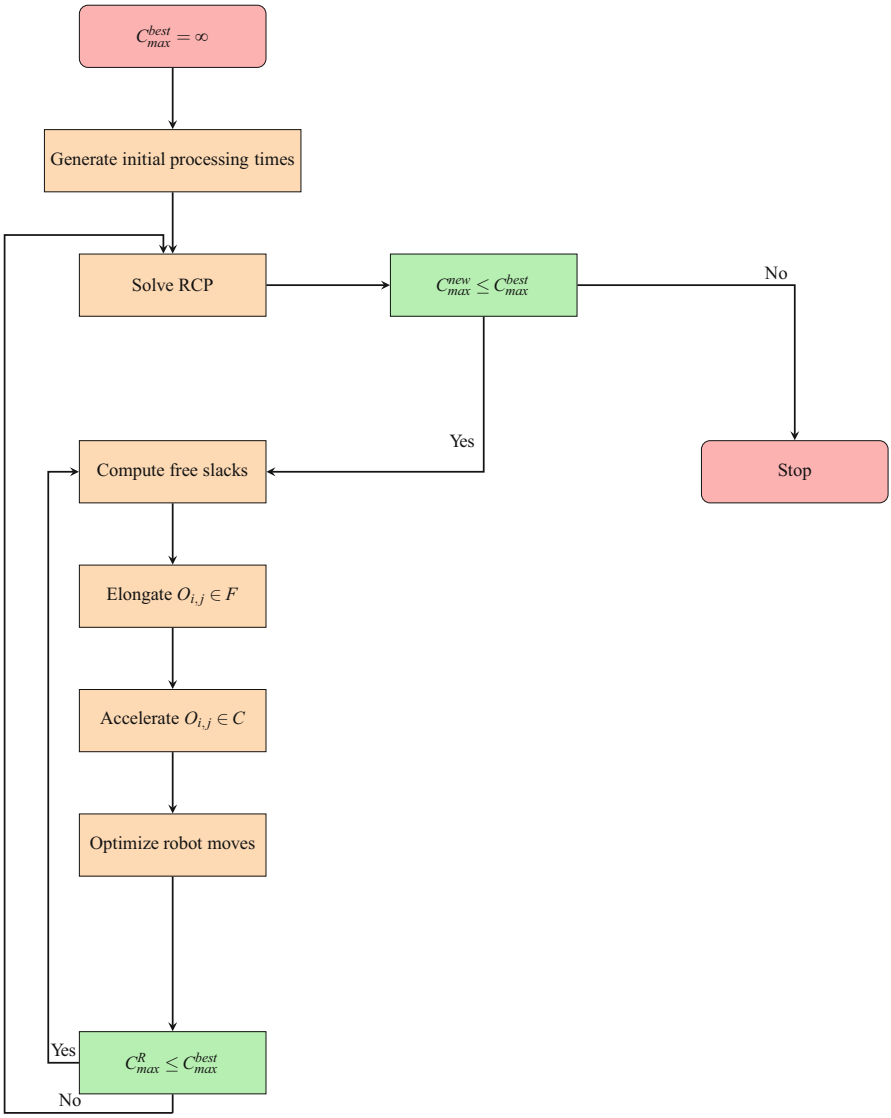


Fig. 4.1 The free-slack-based genetic algorithm

4.4 Computational Study

In this section, we evaluate the performance of this new algorithm against the MIP-based genetic algorithm by Al-Salem et al. (2015). We have coded both genetic algorithms in Microsoft Visual C++ (2010). Also, we used the commercial solver CPLEX 12.6 to solve the proposed MIP formulation and its LP relaxation. All our experiments were run on a Pentium IV 2.4 GHz PC.

The test-bed we have used consists of a large set of randomly generated RCPCPT instances. The number of machines m considered are 3, 4, and 5. For each size, the number of jobs n is taken equal to 10, 12, 14, 16, 18, 20, 30, 40, and 50 jobs. For each combination ($m \times n$), ten instances are generated. The transportation time between a pair of machines M_i and M_k is $2\,|i-k|$ $(i, k = 0, \ldots, m+1)$. The compression rates and the acceleration cost are integers chosen randomly from the interval [1, 10], [1, 5], respectively. The controllable processing times are generated as follows. For each operation O_{ij} $(i = 1, \ldots, m, j \in J)$, the non-compressed processing time $p_{ij}^{\max}$ are drawn from the discrete uniform distributions on [60, 100] and the compressed processing time $p_{ij}^{\min} = p_{ij}^{\max} - a_{ij}c_{ij}$, respectively . Finally, for each instance, the budget was set to $B = \left\lceil 0.5 \sum_{i=1}^{m} \sum_{j \in J} c_{ij} \right\rceil$.

We performed a set of experiments: We solved all small-sized instances ($n = 10, \ldots, 20$) using the MILP formulation after setting the maximum time limit to 3600 s. Then, for each genetic algorithm approach, we computed the mean gap of the ten instances and compared it to the optimal solutions (or the best obtained solution) and with the MIP-based lower bound that is computed after solving a relaxation of the model developed by Al-Salem et al. (2015).

For the free-slack-based genetic algorithm, we set the number of iterations to 10. The initial processing times were generated using the deterministic approach (iteration 1), one by one approach (iterations 2 and 3) and, finally, stochastic approach in the remaining iterations. We applied the stochastic elongation method since this method provided better results.

In order to conduct an efficient comparison with the proposed approach, the MIP-based genetic algorithm was solved under the parameters setting that yields the best performance:

- Population size: 30
- Number of generations: 30
- Maximum number of consecutive generations without improvement before stopping: 5
- Crossover rate: 0.7
- Mutation rate: 0.1

For the RCP genetic algorithm of Kharbeche et al. (2011), we used the following parameter settings:

- Population size: 100
- Number of generations: 100

Table 4.1 Performance of MILP model

m	n	MILP model	
		Time	OPT (%)
3	10	2.92	100
	12	17.45	100
	14	257.08	100
	16	1505.62	80
	18	3600	0
	20	3600	0
4	10	6.46	100
	12	37.14	100
	14	428.42	100
	16	2391.64	60
	18	3600	0
	20	3600	0
5	10	14.24	100
	12	100.21	100
	14	1089.18	80
	16	3137.72	30
	18	3600	0
	20	3600	0

- Maximum number of consecutive generations without improvement before stopping: 20
- Crossover rate: 0.9
- Mutation rate: 0.1

Table 4.1 illustrates the average CPU times for the small-sized instances as well as the percentage of optimal solutions obtained by solving the exact MILP formulation within a CPU time limit of 1 h. Obviously, the exact formulation cannot solve instances beyond 18 jobs.

Table 4.2 reports the CPU time and the average deviations (in %) of small instances against the optimal solutions (we consider only the instances that were optimally solved). We define the deviation from the optimal solution as $\text{GAP.OPT} = \frac{\text{UB}-\text{OPT}}{\text{OPT}}$ as the mean gaps of each combination ($m \times n$) compared with the optimal solution where UB is the obtained solution value by the algorithm and OPT is the optimal solution.

By inspecting Table 4.2, we see that the free-slack-based heuristic outperforms the MIP GA which yields an average deviation of 2.130 % and 1.372 % against 2.195 % and 1.512 % for (3×10) and (5×10), respectively. However, the MIP GA provides better average deviations for the remaining configurations. The gap increases as the number of jobs increases: 3.035 % , 3.060 %, and 2.691 % against 1.899 %, 1.678 %, and 2.490 % for (3×16), (4×16), and (5×16), respectively.

Table 4.2 The average deviation of the proposed heuristics against to optimal solution

m	n	$GAP.OPT_{MIP\ GA}$ (%)	$GAP.OPT_{Free-slack\ GA}$ (%)
3	10	2.130	2.195
	12	2.083	1.437
	14	2.600	1.927
	16	3.035	1.899
	18	–	–
	20	–	–
4	10	1.984	1.574
	12	2.003	2.122
	14	3.250	2.126
	16	3.060	1.678
	18	–	–
	20	–	–
5	10	1.372	1.512
	12	2.063	1.936
	14	1.985	2.393
	16	2.691	2.490
	18	–	–
	20	–	–

The computational times of all instances are displayed in Table 4.3. Clearly, the results show that the free-slack-based algorithm outperforms the MIP-based genetic algorithm for both small and large size instances in terms of CPU times. From Table 4.3, we notice that the computational time required by this heuristic is significantly smaller. In fact, the mean computational times for 3, 4, and 5 machines instances are equal to 12.61, 21.10, and 37.20 s for the MIP-based heuristic against 3.27, 5.39, and 15.82 s for the free-slack-based genetic algorithm.

In Table 4.4, we report the gaps provided by each heuristic compared to the MIP-based lower bound by Al-Salem et al. (2015) for all sizes. The results show a strong evidence of the performance of the proposed procedure. For large sizes, the difference in gaps increases such that for three machines, the average gap is equal to 7.78 %, 8.53 %, and 8.30 % versus 4.45 %, 5.01 %, and 5.18 % for 30, 40, and 50 jobs, respectively. It is worthwhile to observe that although the free-slack GA yielded large deviations for five machines of 7.24, 8.32, and 8.38 %, it still gives better results than the MIP-based GA which resulted a gap of 11.39 %, 11.99 %, and 11.55 % for 30, 40, and 50 jobs, respectively. All data and solutions for all instances are available at www.SchedulingResearch.com.

Table 4.3 CPU times of the genetic algorithms

m	n	MIP GA (s)	Free-slack GA (s)
3	10	3.5535	0.9772
	12	3.4055	1.3365
	14	5.0901	1.4279
	16	7.0677	1.9423
	18	8.1998	2.2263
	20	8.8584	2.4478
	30	20.7665	4.0856
	40	25.0702	6.3721
	50	31.548	8.6826
4	10	5.5975	1.3785
	12	8.2375	2.1676
	14	9.8908	2.8501
	16	15.9007	3.1022
	18	20.2697	3.7577
	20	22.0834	4.6646
	30	28.6515	7.5165
	40	39.5463	9.968
	50	39.7466	13.1611
5	10	11.244	2.8828
	12	18.6268	4.6837
	14	19.8786	5.6156
	16	25.9176	6.3623
	18	21.85	6.9095
	20	29.7981	7.196
	30	32.0577	12.3572
	40	71.4943	29.4042
	50	103.9706	66.9796

4.5 Conclusion

In this chapter, we presented a new free-slack-based genetic algorithm for the RCPCPT. The new approach consists of generating an initial solutions (a set of fixed processing times) then, improve the solution by finding the best sequence of jobs (through an RCP genetic algorithm). Under this sequence, we update the processing times (by elongating the extensible operations and by dividing the earned budget to accelerate the critical ones). The computational study showed the good performance of the proposed heuristic compared to the MIP-based genetic algorithm. The experiments were carried out on randomly generated instances with small and large sizes. For 180 small size instances, The free-slack-based genetic algorithm obtained better solutions than the MIP-based one in 65 % of instances and the latter exhibited better performance in 35 % of the instances. Furthermore, for large sizes (30, 40, and 50), the free-slack approach became the most practical for the 100 % of instances.

Table 4.4 The mean gaps compared to the MIP-based lower bound

m	n	$GAP.LB_{MIP\ GA}$ (%)	$GAP.LB_{Free-slack\ GA}$ (%)
3	10	4.35	4.41
	12	4.13	3.46
	14	4.78	4.09
	16	5.00	3.68
	18	5.27	4.32
	20	5.63	4.14
	30	7.78	4.45
	40	8.53	5.01
	50	8.30	5.18
4	10	4.95	4.53
	12	4.77	4.89
	14	6.36	5.20
	16	5.54	5.21
	18	5.99	5.16
	20	6.20	5.75
	30	9.16	5.81
	40	10.45	6.44
	50	10.05	6.91
5	10	4.63	4.78
	12	5.85	5.73
	14	5.68	6.12
	16	5.86	5.74
	18	7.93	6.58
	20	7.50	6.18
	30	11.39	7.24
	40	11.99	8.32
	50	11.55	8.38

References

Akturk MS, Ilhan T (2011) Single CNC machine scheduling with controllable processing times to minimize total weighted tardiness. Comput Oper Res 38(4):771–781

Al-Salem M, Kharbeche M, Khallouli W, Haouari M (2015) Mip-based heuristics for the robotic cell problem with controllable processing times. Technical Report, CENG/MIE, Qatar University

Biskup D, Cheng TCE (1999) Single-machine scheduling with controllable processing times and earliness, tardiness and completion time penalties. Eng Optim 31(3):329–336

Carlier J, Haouari M, Kharbeche M, Aziz M (2010) An optimization-based heuristic for the robotic cell problem. Eur J Oper Res 202(3):636–645

Dawande MW, Geismar HN, Sethi SP, Chelliah S (2007) Throughput optimization in robotic cells, vol 101. Springer, New York

Janiak A (1991) Single machine scheduling problem with a common deadline and resource dependent release dates. Eur J Oper Res 53(3):317–325

Kharbeche M, Carlier J, Haouari M, Aziz M (2011) Exact methods for the robotic cell problem. Flex Serv Manuf J 23(2):242–261

Koulamas C, Gupta S, Kyparisis GJ (2010) A unified analysis for the single-machine scheduling problem with controllable and non-controllable variable job processing times. Eur J Oper Res 205(2):479–482

Shabtay D, Steiner G (2007) A survey of scheduling with controllable processing times. Discret Appl Math 155(13):1643–1666

Wang J-B, Xia Z-Q (2007) Single machine scheduling problems with controllable processing times and total absolute differences penalties. Eur J Oper Res 177(1):638–645

Xu K, Feng Z, Keliang J (2010) A tabu-search algorithm for scheduling jobs with controllable processing times on a single machine to meet due-dates. Comput Oper Res 37(11):1924–1938

Zdrzałka S (1991) Scheduling jobs on a single machine with release dates, delivery times and controllable processing times: worst-case analysis. Oper Res Lett 10(9):519–523

Chapter 5
Metaheuristic for Randomized Priority Search (Meta-RaPS): A Tutorial

Reinaldo J. Moraga

Abstract This chapter presents a metaheuristic named *Meta*-Heuristic for *Ra*ndomized *P*riority *S*earch (Meta-RaPS), which has been applied to different problems in literature. It mainly explains the overall framework by using an example so the reader can understand how the meta-heuristic works. We at the end identify some future opportunities for research.

Keywords Priority rules • Metaheuristics • Meta-RaPS • Scheduling

5.1 Introduction

Operations research techniques can be utilized to model many practical problems in different industries. Such problems can be formulated as the optimization of a particular decision criterion, expressed as an objective function of some decision variables, restricted by a set of constraints. When the decision variables are discrete, the problem of finding optimal solutions is known as combinatorial optimization. Real-world practice is plenty of examples of combinatorial optimization problems, e.g., the constrained project scheduling problem, the assembly line balancing problem, the vehicle routing and scheduling problem, the facility location problem, the facility layout design problem, the job sequencing and machine scheduling problem, the manpower planning problem, production planning and distribution, and many others. Most of these combinatorial optimization problems are NP-complete, which means that it is unlikely for a polynomial time algorithm that solves such problems optimally to exist.

While the goal of some combinatorial optimization research is to find an algorithm that guarantees an optimal solution in polynomial time as a function of problem size, the main interest in practice is to find a near-optimal or good quality solution in a reasonable amount of time. Numerous approaches to solve

R.J. Moraga (✉)
Department of Industrial and Systems Engineering, Northern Illinois University, 590 Garden Road, Engineering Building 232, DeKalb, IL 60115, USA
e-mail: rmoraga@niu.edu

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_5

combinatorial optimization problems have been proposed, varying from brute-force enumeration through highly esoteric optimization methods. The majority of these methods can be broadly classified as either "exact" algorithms or "heuristic" algorithms. Exact algorithms are those that yield an optimal solution such as the well-known Branch-and-Bound method. The size of practical problems frequently precludes the application of exact methods and thus heuristic algorithms are often used for real-time solution of combinatorial optimization problems.

Heuristic algorithms have, in theory, the chance to find an optimal solution. But, finding the optimal solution can become a remote event because heuristics often get stuck in a local optimal solution. Metaheuristics or modern heuristics deal with this problem by introducing systematic rules to move out of or avoid a local minimum. The common characteristic of these metaheuristics is the use of some mechanisms to escape local optima. Classical and new metaheuristics such as Genetic Algorithms (GA), Simulated Annealing (SA), Tabu Search (TS), Artificial Neural Networks (ANN), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Randomized Search heuristics succeed in leaving the local optimum by temporarily accepting moves which cause a worsening of the objective function value.

Within the randomized search approaches, the Metaheuristic for Randomized Priority Search (Meta-RaPS) and the Greedy Randomized Adaptive Search Procedure (GRASP, developed by Feo and Resende 1989) are two generic, high-level, search procedures that introduce randomness to a construction heuristic as a device to avoid getting trapped at a local optimal solution. Meta-RaPS can be thought as a more general and flexible form of GRASP. Since the author has contributed to the development of the former approach, next sections of this book chapter will focus on Meta-RaPS.

5.2 Meta-RaPS Overview

Meta-RaPS can be described as a generic, high-level strategy used to construct and improve feasible solutions through the utilization of simple heuristic rules used in a randomized fashion. Meta-RaPS was formally introduced by Moraga (2002), but it is based on results of a research conducted by DePuy and Whitehouse (2001) on the application of a modified version of the Computer Method of Sequencing Operations for Assembly Lines (COMSOAL) approach that was developed by Arcus (1966). Meta-RaPS can be thought as a general form of COMSOAL as well as a general form of GRASP, which constructs solutions by introducing randomness to a greedy construction heuristic through the use of a restriction percentage parameter, similar to Meta-RaPS, but it does not give any probabilistic priority to the best alternative considered by the greedy construction heuristic.

Meta-RaPS offers greater flexibility over COMSOAL and GRASP in that it allows user-defined settings of three parameters to provide more efficiency. DePuy et al. (2005) show that the extra flexibility of Meta-RaPS seems beneficial by demonstrating that the Meta-RaPS' parameter settings used to find the best solutions

for the traveling salesman problem (TSP) are different from those settings used to imitate COMSOAL or GRASP. Additionally, Meta-RaPS is reported to have run times not significantly affected by the size of the problem, it is easy to understand and implement, and generate a feasible solution at every iteration.

Meta-RaPS is able to produce high quality solutions for combinatorial optimization problems. Many articles in literature successfully report that fact. For example, the Resource-Constrained Project Scheduling (DePuy and Whitehouse 2001), the Vehicle Routing Problem (Moraga et al. 2001), the TSP (DePuy et al. 2005), the 0-1 Multidimensional Knapsack Problem (Moraga et al. 2005), the Set-Covering Problem (Lan et al. 2007), the Unrelated Parallel Machine Scheduling Problem (Rabadi et al. 2006; Arnaout et al. 2010) the Early/Tardy Single Machine Scheduling Problem (Hepdogan et al. 2009), Parallel Multiple-Area Spatial Scheduling Problem with Release Times (Garcia and Rabadi 2011), Aerial Refueling Scheduling Problem (Kaplan et al. 2011), and Runway combined arrival-departure aircraft sequencing problem (Hancerliogullari et al. 2013).

Some articles in literature also report the use of several features to improve Meta-RaPS procedure. Lan and DePuy (2006) discuss the effectiveness of using memory mechanism, showing promising results. Arin and Rabadi (2012a, b) use path relinking and Q-learning mechanisms within Meta-RaPS framework for solving the 0-1 multidimensional knapsack problem, obtaining very good results. Al-Duoli and Rabadi (2013, 2014) proposed incorporating intelligence into Meta-RaPS via data mining and machine learning techniques, specifically using Inductive Decision Trees. With the incorporation of memory and learning features, not only can Meta-RaPS improvement phase be enhanced, but the construction phase also. In addition, a study on the incorporation of dynamic parameter setting methods within Meta-RaPS has also been reported (Hepdogan et al. 2008) in an attempt to make the approach less dependent on the initial off-line parameter setting efforts. The methods tested are Reactive Search (RS), Ranking and Selection (R&S) and Non-parametric based Genetic Algorithms (NPGA). Most of the previous articles report on Meta-RaPS potential and the need of extending its applications to many other areas.

In the following section, the Meta-RaPS framework will be demonstrated using a simple example for a NP-hard problem.

5.3 Meta-RaPS Framework

Meta-RaPS algorithm attempts to balance the usage of both construction and improvement heuristics to generate feasible solutions (S). It is important to note that most of the other competing metaheuristics make an unbalanced usage of their search time by privileging improvement heuristics at the expense of construction heuristics. For example, standard TS, SA, or GA procedures randomly construct a first solution, or an initial pool of solutions, and the rest of the search time is spent using an improvement procedure.

The execution of Meta-RaPS is controlled by using four parameters: the number of iterations (I), the priority percentage (p), the restriction percentage (r), and the improvement percentage (i). The number of constructed feasible solutions is determined by the number of iterations. In general, a construction heuristic builds a solution by systematically adding feasible items to the current solution. The item with the best priority value is added to the current solution. With both p and r parameters, Meta-RaPS modifies the way a general construction heuristic chooses the next item to add to the solution by occasionally choosing a job that does not have the best priority value. In addition, a solution improvement algorithm can be included in Meta-RaPS by using the i parameter. Once a solution has been constructed, Meta-RaPS may proceed to improve it further through neighborhood search algorithms. The improvement heuristic is performed if the construction phase's solution value is below certain threshold δ, determined by i and the range between both best and worst unimproved solution values found so far. Finally, the best solution from all iterations is reported (S^* and $f(S^*)$).

In general, Meta-RaPS can be summarized using the following pseudo-code, assuming a minimization problem (Fig. 5.1):

In the following sections a short tutorial will be provided on how to apply Meta-RaPS framework to a particular scheduling problem.

I: number of iterations
p: priority percentage
r: restriction percentage
i: improvement percentage
S: current schedule
f(*S*)*: current objective function value*
S: best solution*
f(*S**)*: best objective function value*
B: best constructed solution so far
W: worst constructed solution so far
δ: threshold on the objective function to accept current solution for improvement

```
1:    Set I, p, r, i, S*=∅, and f(S*)=INF; B=inf; W=-inf;
2:    Repeat I times
3:      S = construction (p, r);
4:      Update (B, W);
5:      If f(S) ≤ δ(i, B, W),
6:              S = local_search (S);
7:      End
8:      Update(S*, f(S*));
9:    End
```

Fig. 5.1 Pseudo code for Meta-RaPS framework

5.3.1 The Single Machine Total Weighted Tardiness Problem

In order to show how Meta-RaPS works, the scheduling problem of minimizing the total weighted tardiness on a single machine will be used. The single machine total weighted tardiness problem, $1 \mid\mid \sum_{(j=1)}^{n} w_j T_j$, may be described as a set of n jobs to be processed without preemption on a single machine that can process only one job at a time. To simplify the problem, it is assumed that all jobs are available at time zero. Job j has a processing time p_j, a due date d_j and has a positive weight w_j. In addition, a weighted tardiness penalty is incurred for each time unit of tardiness T_j a job j is completed after its due date d_j. Therefore the problem can be formally sated as: find a schedule S that minimizes the objective function $f(S) = \sum_{j=1}^{n} w_j T_j$.

Consider the following data instance proposed by Pinedo (2012) in exercise 3.10, pg. 64, for a single machine and seven jobs. Numerical values for processing times, weights, and due dates are given in Table 5.1.

Two main steps are necessary to successfully apply Meta-RaPS, and these are to design effective construction and improvement heuristics. For the single machine total weighted tardiness problem, a very simple and well-known construction heuristic will be used in order to demonstrate how a simple heuristic evolves into an effective one with the collaboration of Meta-RaPS mechanisms.

5.3.2 Heuristics for the $1 \mid\mid \sum_{j=1}^{n} w_j T_j$ Problem

The problem used as an example in this book chapter has been extensively studied. Lawler (1977) shows that the $1 \mid\mid \sum_{j=1}^{n} w_j T_j$ problem is strongly NP-hard. Thus, various heuristic methods can be found in literature to obtain good solutions. In general, all these methods can be broadly classified as construction and improvement heuristics.

5.3.2.1 Construction Heuristic

There are some classical commonly used heuristic rules for the $1 \mid\mid \sum_{j=1}^{n} w_j T_j$ problem. For instance, some of these rules are: Cost Over Time (COVERT), Largest Penalty per Unit Length (LPUL), Shortest Weighted Processing Time (WSPT),

Table 5.1 Data for numerical example

j	1	2	3	4	5	6	7
p_j	6	18	12	10	10	17	16
w_j	1	5	2	4	1	4	2
d_j	8	42	44	24	90	85	68

S: current schedule
f(*S*)*: current objective function value*
J^a*: set of unscheduled jobs*
j: job with minimum due date*
d_j*: due date of job j*
k: position in schedule S

1:	Set $S=\varnothing$, $J^a=\{1,...,n\}$ and $k=1$.
2:	Let j^* denote the job that satisfies $d_{j^*} = \min_{j \in J^a}(d_j)$, ties are broken arbitrarily.
3:	Assign job j^* to S in position k and delete j^* from J^a.
4:	If $J^a=\varnothing$, then STOP, report S and $f(S)$; otherwise set $k=k+1$ and go to line 2.

Fig. 5.2 Pseudo code for Earliest Due Date rule

and Critical Ratio (CR)—see Sule (2007) for a detailed explanation of them. The simplest one for this problem found in literature is the Earliest Due Date (EDD) rule, which prioritizes jobs according to their *due date*. Out of the pool of unscheduled jobs, the one with the *earliest due date* is scheduled first. The EDD rule is described in Fig. 5.2.

5.3.2.2 Improvement Heuristic

A well-known local search technique to improve constructed solutions in scheduling problems is the Adjacent Pairwise Interchange (API). Given the original constructed schedule S, where job i starts its processing at time t followed by job j, the API procedure consists of forming a new schedule S'' where job j starts at time t and is followed by i. All other jobs remind in their original position. The total weighted tardiness of jobs processed before and after jobs i and j is not affected by the interchange. The difference in the values of the total weighted tardiness under schedules S and S'' is only due to jobs i and j. The API procedure is described in Fig. 5.3.

Meta-RaPS will use both EDD and API at each iteration, attempting to balance the use of both while searching for a good solution.

5.3.3 Meta-RaPS Application

During the construction phase, Meta-RaPS uses the EDD rule as its basic procedure. However, while adding jobs to the solution, the p parameter is used to determine the percentage of time the next job added to the solution has the best priority value (d_{j^*}) as in line 2. The remaining time ($1 - p/100$), the next job added to the solution

S: current schedule
f(S): current objective function value
S': auxiliary schedule
s: position in schedule S and S'
n: number of jobs

```
5:    Set S'=S and s=1.
6:    While s < n,
7:        Get jobs i and j in S from positions s and s+1, respectively.
8:        Place jobs j and i in S' at positions s and s+1, respectively.
9:        If f(S') ≤ f(S); S=S'.
10:       s=s+1.
11:   End
12:   Report S and f(S).
```

Fig. 5.3 Pseudo code for Adjacent Pairwise Interchange rule

```
2':   Generate a random number rnd between 0 and 1.
      If rnd ≤ p,
      Then, j* is the job that satisfies d_j* = min_{j∈J^a}(d_j), ties are broken arbitrarily.
      Else, take j* randomly from the set of all j∈J^a that satisfy
            d_j ≤ α +(β - α)r, where α = min_{j∈J^a}(d_j), β = max_{j∈J^a}(d_j)
      End
```

Fig. 5.4 Pseudo code for EDD modified by Meta-RaPS

is randomly chosen from those unscheduled jobs whose d_j values are within an r percentage out of the range of due dates above the d_{j*} value, which further modifies line 2. All the other lines remain the same. Therefore, line 2 becomes line 2′ as shown in Fig. 5.4:

The rationale in line 2′ takes advantage of two experimental findings. First, if some degree of randomness is introduced into a particular heuristic (or priority rule), by using r, the corresponding results improve dramatically. Second, if two heuristics are randomly combined, by using p, the solution obtained usually outperforms the solution of either heuristic individually (DePuy et al. 2005).

Once a schedule S has been constructed, the i parameter is used to decide whether S will go through the improvement phase or not. In order to accomplish this, while Meta-RaPS is being executed, it keeps track of the best (B) and worst (W) total weighted tardiness values of schedules constructed in order to calculate a threshold, $\delta(i) = B + (W - B)i/100$. If $f(S) \leq \delta(i)$, then S goes through the improvement stage. The underlying rationale for this strategy is the expectation that good unimproved solutions lead to better neighboring solutions. The improvement phase in this case consists of the API procedure described previously.

The Meta-RaPS algorithm for the single machine total weighted tardiness problem is the following (Fig. 5.5):

t: counter of the number of iterations
rnd: random number between 0 and 1
S'': auxiliary schedule
α: minimum due date out of all unscheduled jobs, $\alpha = \min_{j \in J^a}(d_j)$
β:maximum due date out of all unscheduled jobs, $\beta = \max_{j \in J^a}(d_j)$

```
1:    Set parameters I, p, r, and i; S*=∅, f(S*)=inf; B=inf; W=-inf;
2:    For t = 1 to I,
3:       Set S=∅, J^a={1,..., n} and k=1;
4:       Generate a random number rnd;
5:       If  rnd ≤ p/100,
6:       Then, j*is the job that satisfies d_j* = min_{j∈J^a}(d_j) ; ties are broken arbitrarily.
7:       Else, take j* randomly from the set of all j∈J^a that satisfy d_j ≤ α +(β - α)r/100;
8:       End
9:       Assign job j* to S in position k, set k=k+1 and delete j* from J^a;
10:      If J^a=∅, Then go to line 11, Else go to line 4; End
11:      Update (B); Update(W);
12:      If f(S) ≤ B+(W-B)i/100,
13:         s=1;
14:         While s < n,
15:            Set S''=S;
16:            Get jobs i and j in S from positions s and s+1, respectively;
17:            Place jobs j and i in S' at positions s and s+1, respectively;
18:            If f(S'') ≤ f(S), S'=S'';
19:            s=s+1.
20:         End
21:         Set S=S';
22:      End
23:      Update (S*, f(S*));
24:   End
25:   Report S* and f(S*);
```

Fig. 5.5 Pseudo code for solving the single machine total weighted tardiness problem with Meta-RaPS

5.3.4 Numerical Example

In this section the data from Table 5.1 will be used to show the calculations for the first iteration only.

In line 1, the values for Meta-RaPS are assumed.

1: $I = 5, p = 20, r = 50, i = 90; S^* = \varnothing, f(S^*) = \text{inf}, B = \text{inf}, W = \text{-inf}$
2: For iteration, $I = 1$:
3: Set $S = \varnothing$, $J^a = \{1, 2, 3, 4, 5, 6, 7\}$ and $k = 1$;

From lines 4 to 10, the first job to be schedule is selected. These lines are repeated for n times until all jobs are assigned to the schedule.

4: Generate a random number, $rnd = 0.56$;
5: Since rnd is greater than 20/100, go to line 7.
7: Therefore, out of all jobs that belong to J^a and whose due dates satisfy the condition $d_j \leq 8 + (90-8)50/100 = 49$, job 2 is randomly selected. Go to line 9.
9: Assign job 2 to S in position 1, set $k = 2$ and delete job 2 from J^a; so, $S = \{1\}$, $J^a = \{1,3,4,5,6,7\}$.
10: Since $J^a \neq \varnothing$, go to line 4.
4: Generate a random number, $rnd = 0.17$;
5: Since $rnd < 20/100$, go to line 6.
6: Select Job 1 since it has the minimum due date value out of all jobs in J^a. Go to line 9.
9: Assign job 1 to S in position 2, set $k = 3$ and delete job 1 from J^a; so, $S = \{2,1\}$, $J^a = \{3,4,5,6,7\}$.
10: Since $J^a \neq \varnothing$, go to line 4.
4: Generate a random number, $rnd = 0.79$;
5: Since $rnd \geq 20/100$, go to line 7.
7: Job 3 is randomly picked out of all $j \in J^a$ whose due dates satisfy $d_j \leq 24 + (90-24)50/100 = 57$. Go to line 9.
9: Assign job 3 to S in position 3, set $k = 4$ and delete job 3 from J^a; so, $S = \{2,1,3\}$, $J^a = \{4,5,6,7\}$.
10: Since $J^a \neq \varnothing$, go to line 4.
4: Generate a random number, $rnd = 0.48$;
5: Since $rnd \geq 20/100$, go to line 7.
7: Job 4 is the only $j \in J^a$ whose due date satisfy $d_j \leq 24 + (90-24)50/100 = 57$. Go to line 9.
9: Assign job 4 to S in position 4, set $k = 5$ and delete job 4 from J^a; so, $S = \{2,1,3,4\}$, $J^a = \{5,6,7\}$.
10: Since $J^a \neq \varnothing$, go to line 4.
4: Generate a random number, $rnd = 0.16$;
5: Since $rnd < 20/100$, go to line 6.
6: Select Job 7 since it has the minimum due date value out of all jobs in J^a. Go to line 9.
9: Assign job 7 to S in position 5, set $k = 6$ and delete job 7 from J^a; so, $S = \{2,1,3,4,7\}$, $J^a = \{5,6\}$.
10: Since $J^a \neq \varnothing$, go to line 4.
4: Generate a random number, $rnd = 0.40$;
5: Since $rnd \geq 20/100$, go to line 7.
7: Job 6 is randomly picked out of all $j \in J^a$ whose due dates satisfy $d_j \leq 85 + (90-85)50/100 = 87.5$. Go to line 9.
9: Assign job 6 to S in position 6, set $k = 7$ and delete job 6 from J^a; so, $S = \{2,1,3,4,7,6\}$, $J^a = \{5\}$.

10: Since $J^a \neq \varnothing$, go to line 4.
4: Generate a random number, $rnd = 0.42$;
5: Since $rnd \geq 20/100$, go to line 7.
7: Job 5 is the only remaining job $j \in J^a$. Go to line 9.
9: Assign job 5 to S in position 7, set $k = 8$ and delete job 5 from J^a; so, $S = \{2,1,3,4,7,6,5\}$, $J^a = \varnothing$.
10: Since $J^a = \varnothing$, go to line 11.

Meta-RaPS constructed the first schedule S by assigning all jobs. In line 11, *Best* and *Worst* unimproved solutions are updated.

11: Since $f(S) = 104 < B$, $B = 104$; Since $f(S) = 104 > W$, $W = 104$.

Using the values of B and W, Meta-RaPS evaluates in line 12 whether the current constructed solution will go through the improvement phase.

12: If $f(S) \leq 104 + (104\text{-}104)90/100 = 104$, the solution goes through improvement.
13: $s = 1$.

Lines 14–19 are repeated six times $(n-1)$ until all the API moves are performed. At each move, the original constructed schedule is used and the best solution reported at the end.

14: While $s = 1 < 7$,
15: Set $S' = S$;
16: Get jobs 2 and 1 in S from positions 1 and 2, respectively;
17: Put jobs 1 and 2 in S'' at positions 1 and 2, respectively; $S'' = \{1,2,3,4,7,6,5\}$.
18: Since $f(S'') = 88 < f(S) = 104$, $S' = S'' = \{1,2,3,4,7,6,5\}$;
19: $s = 2$; go to line 14.
14: While $s = 2 < 7$,
15: Set $S'' = S$;
16: Get jobs 1 and 3 in S from positions 2 and 3, respectively;
17: Put jobs 3 and 1 in S'' at positions 2 and 3, respectively; $S'' = \{2,3,1,4,7,6,5\}$.
18: Since $f(S'') = 116 > f(S) = 104$;
19: $s = 3$; go to line 14.
14: While $s = 3 < 7$,
15: Set $S'' = S$;
16: Get jobs 3 and 4 in S from positions 3 and 4, respectively;
17: Put jobs 4 and 3 in S'' at positions 3 and 4, respectively; $S'' = \{2,1,4,3,7,6,5\}$.
18: Since $f(S'') = 60 < f(S) = 104$; $S' = S'' = \{2,1,4,3,7,6,5\}$;
19: $s = 4$; go to line 14.
14: While $s = 4 < 7$,
15: Set $S'' = S$;
16: Get jobs 4 and 7 in S from positions 4 and 5, respectively;
17: Put jobs 7 and 4 in S'' at positions 4 and 5, respectively; $S'' = \{2,1,3,7,4,6,5\}$.
18: Since $f(S'') = 168 > f(S) = 104$;
19: $s = 5$; go to line 14.

14: While $s = 5 < 7$,
15: Set $S'' = S$;
16: Get jobs 7 and 6 in S from positions 5 and 6, respectively;
17: Put jobs 6 and 7 in S'' at positions 5 and 6, respectively; $S'' = \{2,1,3,4,6,7,5\}$.
18: Since $f(S'') = 126 > f(S) = 104$;
19: $s = 6$; go to line 14.
14: While $s = 6 < 7$,
15: Set $S'' = S$;
16: Get jobs 6 and 5 in S from positions 6 and 7, respectively;
17: Put jobs 5 and 6 in S'' at positions 6 and 7, respectively; $S'' = \{2,1,3,4,7,5,6\}$.
18: Since $f(S'') = 120 > f(S) = 104$;
19: $s = 7$; go to line 21.

Line 21 updates the schedule S with that obtained from the API procedure. Line 23 updates the best solution obtained so far.

21: Set $S = S' = \{2,1,4,3,7,6,5\}$; go to line 23.
23: If $f(S) < f(S^*)$, $f(S^*) = f(S)$, $S^* = S$; go to line 2 to continue with next iteration.

Therefore, after the first iteration, the constructed solution is {2,1,3,4,7,6,5} with a total weighted tardiness of 104. Due to the fact that this solution goes through the improvement phase, the improved schedule is {2,1,4,3,7,6,5} with a total weighted tardiness of 60, which is also a solution obtained after the improvement updates the best objective value and schedule. Table 5.2 shows the summary of the rest of the iterations and the evolution of total weighted tardiness value is shown in Fig. 5.6. The best solution found (S^*) out of five iterations is {1,4,2,3,7,6,5} and the total weighted tardiness value is 4.

In this example, values for p, r, and i were assumed, but do not necessarily represent the best parameter setting. Therefore, additional steps should be made in order to find the best parameter setting to run Meta-RaPS. For a good discussion on different techniques, the reader is referred to Hepdogan et al. (2008).

Table 5.2 Summary of the iterations, I, using Meta-RaPS

	Construction					Improvement				
I	S	$f(S)$	B	W	$\delta(i)$	$f(S) \leq \delta(i)$?	S	$f(S)$	S^*	$f(S^*)$
–	–	–	inf	-inf	–	–	–	–	∅	inf
1	{2,1,3,4,7,6,5}	104	104	104	104	Yes	{2,1,4,3,7,6,5}	60	{2,1,4,3,7,6,5}	60
2	{1,3,4,2,7,6,5}	36	36	104	97.2	Yes	{1,4,3,2,7,6,5}	20	{1,4,3,2,7,6,5}	20
3	{3,1,4,2,7,6,5}	46	36	104	97.2	Yes	{1,3,4,2,7,6,5}	36	{1,4,3,2,7,6,5}	20
4	{4,1,2,3,7,6,5}	12	12	104	94.8	Yes	{1,4,2,3,7,6,5}	4	{1,4,2,3,7,6,5}	4
5	{2,3,4,1,7,6,5}	102	12	104	94.8	No	–	–	{1,4,2,3,7,6,5}	4

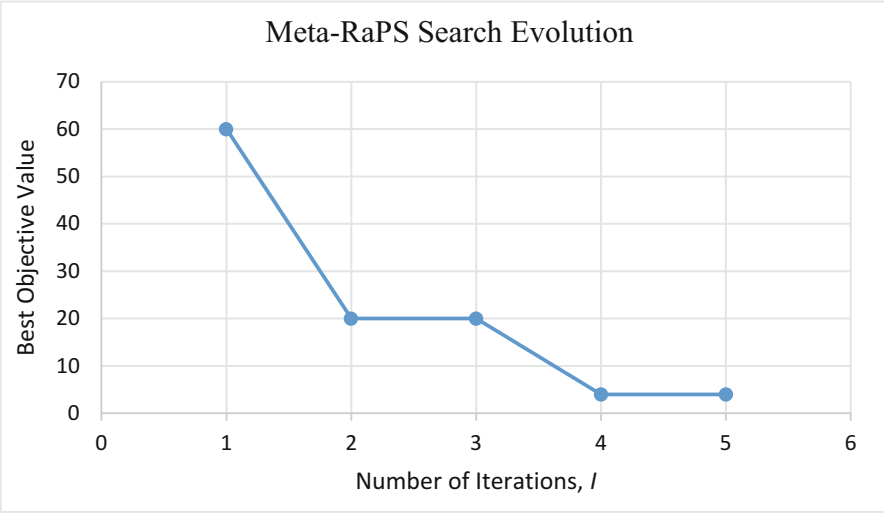


Fig. 5.6 Evolution of the objective function value for $I = 5$

5.4 Conclusions and/or Remarks

As shown in the example of this book chapter, Meta-RaPS framework is very easy to understand and to implement. Its procedure has already been successfully applied to different NP-hard problems. This demonstrates that the inclusion of randomness in a priority scheme performs significantly better than the priority rule itself. Meta-RaPS produces near-optimal solutions. This is important because the ultimate goal of practitioners is to have methods that produce near-optimal solutions in a reasonable amount of time rather than guaranteeing optimal ones after a long computational time.

However, it would be desirable to extend Meta-RaPS framework to another class of problems, such as: multi-objective combinatorial optimization and large-scale optimization problems. This would be important because most of the problems found in real life belong to the category of large-scale optimization problems. These problems normally involve a large number of continuous and/or discrete variables. As a consequence of new algorithmic developments and of the increased power of computers, a growing research community is being attracted by large-scale optimization problems. Meta-RaPS application to this type of problems may shed some light on its real advantages as well as on more precise generalization of the approach as an all-purpose strategy.

The incorporation of memory and learning mechanisms would constitute a vital enhancement for Meta-RaPS framework. It is important to remark that most

of the modern metaheuristics use memory and learning mechanism during the improvement phase. It would, however, be desirable to advance Meta-RaPS by incorporating such mechanisms during the construction phase.

References

Al-Duoli F, Rabadi G (2013) Employing learning to improve the performance of Meta-RaPS. Procedia Comput Sci 20: 46–51. Complex Adaptive Systems, Baltimore, MD, 13–15 November 2013

Al-Duoli F, Rabadi G (2014) Data mining based hybridization of MetaRaPS. In: Proceedings of complex adaptive systems conference, Philadelphia, 3–5 November 2014

Arcus A (1966) COMSOAL: a computer method of sequencing operations for assembly lines, I the problem in simple form. In: Buffa E (ed) Readings in production and operations management Wiley & Sons, New York

Arin A, Rabadi G (2012a) Meta-RaPS with path relinking for the 0-1 multidimensional knapsack problem. In: IS'2012 - 2012 6th IEEE international conference intelligent systems, Proceedings, 347–352, Sofia, Bulgaria, September 6-8, 2012

Arin A, Rabadi G (2012b) Meta-RaPS with Q Learning approach intensified by path relinking for the 0-1 multidimensional knapsack problem. In: Proceedings of the 14th International Conference on Artificial Intelligence (ICAI'12), Las Vegas, 16–19 July 2012

Arnaout JP, Rabadi G, Musa R (2010) A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. J Intell Manuf 21(6):693–701

DePuy G, Whitehouse G (2001) A simple and effective heuristic for the multiple resource allocation problem. Int J Prod Res 32(4):24–31

DePuy GW, Moraga RJ, Whitehouse GE (2005) Meta-RaPS: a simple and effective approach for solving the traveling salesman problem. Transp Res Part E Logist Transp Rev 41(2):115–130

Feo TA, Resende MGC (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8:67–71

Garcia C, Rabadi G (2011) A Meta-RaPS algorithm for spatial scheduling with release times. Int J Planning Scheduling 1(1/2):19–31

Hancerliogullari G, Rabadi G, Al-Salem AH, Kharbeche M (2013) Greedy algorithms and metaheuristics for a multiple runway combined arrival-departure aircraft sequencing problem. J Air Transp Manag 32:39–48

Hepdogan S, Moraga R, DePuy GW, Whitehouse GE (2008) Nonparametric comparison of two dynamic parameter setting methods in a meta-heuristic approach. J Syst Cybern Inform 5(5):46–52

Hepdogan S, Moraga R, DePuy GW, Whitehouse GE (2009) A Meta-RaPS solution for the early/tardy single machine scheduling problem. Int J Prod Res 47(7):1717–1732

Kaplan S, Rabadi G (2012a) Exact and heuristic algorithms for the aerial refueling parallel machine scheduling problem with due date-to-deadline window and ready times. Comput Ind Eng 62(1):276–285. doi:10.1016/j.cie.2011.09.015

Kaplan S, Rabadi G (2012b) Simulated annealing and metaheuristic for randomized priority search algorithms for the aerial refueling parallel machine scheduling problem with due date-to-deadline windows and release times. Eng Optim (April 2013), 1–21

Kaplan S, Arin A, Rabadi G (2011) Meta-RaPS algorithm for the aerial refueling scheduling problem. 175–191. Retrieved from http://hdl.handle.net/2060/20110012105

Lan G, DePuy GW (2006) On the effectiveness of incorporating randomness and memory into a multi-start metaheuristic with application to the set covering problem. Comput Ind Eng 51:362–374

Lan G, DePuy GW, Whitehouse GE (2007) An effective and simple heuristic for the set covering problem. Eur J Oper Res 176(3):1387–1403

Lawler EL (1977) A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. Ann Discret Math 1(C):331–342

Moraga RJ (2002) Meta-RaPS: an effective solution approach for combinatorial problems. Ph.D. Dissertation, University of Central Florida, FL 32816, USA

Moraga RJ, Whitehouse GE, DePuy GW, Neyveli B, Kuttuva S (2001) Solving the vehicle routing problem using the Meta-RaPS approach. In: 29th international conference on computers and industrial engineering, Montreal, 1–3 November

Moraga RJ, DePuy GW, Whitehouse GE (2005) Meta-RaPS approach for the 0-1 multidimensional knapsack problem. Comput Ind Eng 48(2):83–96

Pinedo M (2012) Scheduling: theory, algorithms, and systems, 4th edn. Springer, New York

Rabadi G, Moraga R, Al-Salem A (2006) Heuristics for the unrelated parallel machine scheduling problem with setup times. J Intell Manuf 17:85–97

Sule D (2007) Production planning and industrial scheduling: examples, case studies and applications, 2nd edn. Taylor & Francis CRC Press, Boca Raton

Chapter 6
Performance of an Intensification Strategy Based on Learning in a Metaheuristic: Meta-RaPS with Path Relinking

Arif Arin and Ghaith Rabadi

Abstract Intensification and diversification in metaheuristics are two main strategies to enhance the search process and solution quality. In Meta-RaPS (Metaheuristic for Randomized Priority Search), a recent memoryless metaheuristic, intensification and diversification strategies are controlled only by the level of randomness specified by its parameters. We introduce in this paper a Path Relinking (PR) learning algorithm and integrate it into Meta-RaPS to intelligently enhance its intensification capability by learning "good" attributes of the best solutions. To evaluate its performance, the proposed Meta-RaPS PR is tested on the 0-1 Multidimensional Knapsack Problem (MKP). The results show that applying PR as an intensification strategy in Meta-RaPS is very effective as it outperformed other approaches used in the literature with this problem. The PR approach also transformed the memoryless nature of Meta-RaPS into an "intelligent" algorithm.

Keywords Intensification • Path relinking • Meta-RaPS • Metaheuristics • Knapsack problem

6.1 Introduction

Metaheuristics such as Genetic Algorithms (GA), Simulated Annealing (SA), Tabu Search (TS), and Ant Colony Optimization (ACO) have generally shown to be effective at solving difficult optimization problems due to their ability to guide

A. Arin (✉)
Department of Business Administration, Middle East Technical University, Universiteler Mahallesi, Dumlupinar Bulvari No:1, 06800 Ankara, Turkey
e-mail: arif.arin@metu.edu.tr

G. Rabadi
Engineering Management & Systems Engineering, Old Dominion University, Engineering Systems Building, Room 2102, Norfolk, VA 23529, USA
e-mail: grabadi@odu.edu

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_6

the search out of local optima. Such algorithms are intelligent in the sense that they store and learn from information related to the search history to reach high quality solutions. Learning in metaheuristics is achieved mainly through storing information in memory during the search and extracting it later to better guide the search. Memory and learning in metaheuristics vary from one metaheuristic to another (Arin and Rabadi 2012). While the tabu list represents memory in TS, in most metaheuristics such as Evolutionary Algorithms (EA) and GA, the search memory is limited to a population of solutions. In ACO, the pheromone matrix is the main memory component to aid the search, whereas in Estimation Distribution Algorithms (EDA), it is a probabilistic learning model that composes the search memory.

Memorized data are not only raw input; they could be distribution information that pertains to the solutions. This is emphasized by Dréo et al. (2007) who presented the following Adaptive Learning Search (ALS) algorithm:

1. Initialize a sample.
2. Until stopping criteria are met, do:

   (a) Sampling: either explicit, implicit, or direct;
   (b) Learning: the algorithm extracts information from the sample;
   (c) Diversification: the algorithm searches for new solutions;
   (d) Intensification: the algorithm searches to improve the existing sample;
   (e) Replacement: the algorithm replaces the previous sample with the new one.

As can be seen in ALS, intensification and diversification are two important strategies for the memory and learning structure. According to Rochat and Taillard (1995), "diversification drives the search to examine new regions, and intensification focuses more intently on regions previously found to be good." Intensification strategies modify the algorithm to search promising regions more thoroughly based on high quality solution features found during the search process of the neighborhood of elite solutions, or by modifying choice rules to favor the inclusion of attributes of these solutions. On the other hand, diversification strategies encourage the algorithm to explore new regions and update its long-term memory via specialized mechanisms. Local search often relies on diversification strategies to reach better solutions and help prevent cycling of the search process, which makes the algorithm more robust.

The main difficulty for the search in metaheuristics is balancing the intensification and diversification strategies. The search process can prematurely converge towards a local optimum unless the search process can be diversified by increasing the number of moves or components that are labeled as undesirable. In Meta-RaPS (Meta-heuristic for Randomized Priority Search), which has shown very promising performance despite being classified as a memoryless metaheuristic, intensification and diversification strategies are controlled by controlling the level of randomness as there is no mechanism to memorize the information created in the solution process, nor is there a mechanism to learn the structure of this process in making future decisions. It, however, has an improvement phase in

which local search techniques are used to improve promising solutions. Therefore, we propose incorporating a learning mechanism into Meat-RaPS to improve its efficiency in obtaining high quality solutions based on Path Relinking (PR) approach in which "good" attributes of the best solutions are memorized while the algorithm is progressing. The approach is named Path Relinking because it generates a path between two solutions called initial and guiding solutions, to create new solutions. While progressing, the initial solution gradually transforms to the guiding solution by incorporating the attributes of the guiding solution.

To evaluate the performance of the proposed approach, the 0-1 Multidimensional Knapsack Problem (MKP), will be used as a test bed. The 0-1 MKP is the generalized form of the classical knapsack problem (KP). In KP there is a knapsack with an upper weight limit b, and a set of n items with different profits c_j and weights a_j per item j. The problem is to select the items from the set of items such that the total profit of the selected items is maximized without violating the upper weight limit constraint of the knapsack b. If m knapsacks exist, the problem becomes the MKP in which each knapsack has a different upper weight limit b_i, and an item j has a different weight a_{ij} for each knapsack i. The problem can be formulated as follows:

$$\text{Maximize} \quad \sum_{j=1}^{n} c_j x_j \tag{6.1}$$

$$\text{Subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m; \quad j = 1, \ldots, n \tag{6.2}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n \tag{6.3}$$

where x is a vector of binary variables such that $x_j = 1$ if item j is selected, and $x_j = 0$ otherwise. The MKP can be accepted as a special case of the general linear 0-1 integer programming problem with nonnegative coefficients (Martello and Toth, 1990). In the literature it is assumed that profits, weights, and capacities are positive integers. However they can be easily extended to the case of real values (Martello and Toth 1990). This is a resource allocation problem, which can be used to model many problems in the literature such as the capital budgeting, project selection, cutting stock, and many loading problems. Although the classical KP is weakly NP-hard, the MKP is much more difficult (NP-hard) even for $m = 2$ (Garey and Johnson 1979).

6.2 Meta-RaPS

Meta-RaPS was initially based on the work by DePuy and Whitehouse (2001) as a result of the research conducted on the application of a modified version of

COMSOAL (Computer Method of Sequencing Operations for Assembly Lines) approach that was developed by Arcus (1966). Meta-RaPS was then formally introduced by Moraga (2002) who defines it as a generic, high-level strategy used to modify greedy algorithms based on the insertion of a random element, which integrates priority rules, randomness, and sampling. Indeed, Meta-RaPS is a general form of GRASP (greedy randomized adaptive search procedure), and although GRASP generates solutions by introducing randomness, it does not implement any probabilistic priority to the best solutions.

Meta-RaPS is a two-phase metaheuristic: a constructive phase to create feasible solutions and an improvement phase to improve them. The algorithm can be controlled by four parameters: number of iterations (I), the priority percentage ($p\%$), the restriction percentage ($r\%$), and the improvement percentage ($i\%$). In constructing a solution, Meta-RaPS does not select the component or activity with the best priority value in every iteration, nor does it select the one with the lowest incremental cost. Instead, it may randomly accept an activity or component with a good priority value, but not necessarily the best one to avoid falling in local optima. The parameter $p\%$ is used to decide the percentage of time a component or activity with the best priority value will be added to the current partial solution, and $100\,\% - p\%$ of the time it will be randomly selected from a candidate list (CL) that contains "good" components or activities. This decision is made by taking a random number (rand) between 0 and 1, then comparing it with the parameter $p\%$. If the random number is smaller than or equal to $p\%$ then the algorithm selects the component or activity with the best priority value, else it selects from the CL, which is created by including items whose priority values are within $r\%$ of the best priority value. In the construction phase, the level of the randomness is adjusted by controlling the values of the parameters $p\%$ and $r\%$ where smaller values of $p\%$ and larger values of $r\%$ will randomize (diversify) the search more. The construction phase is completed when a feasible solution is produced. The improvement phase is performed only if the feasible solutions generated in the construction phase are within $i\%$ of the best unimproved solution value from the preceding iterations. The rationale here is to improve only promising solutions and not to waste time on inferior solutions. Pseudo code for Meta-RaPS is shown in Fig. 6.1.

DePuy et al. (2001) emphasized that the advantages of the Meta-RaPS over other metaheuristics are that its run times are not significantly affected by the size of the problem, it is easy to understand and implement, and can generate a feasible solution at every iteration. The simple nature of Meta-RaPS coupled with its ability to generate high quality solutions, makes it a good metaheuristic method for combinatorial optimization problems.

Meta-RaPS produces high quality solutions when applied to discrete optimization problems, such as the Resource Constrained Project Scheduling Problem (DePuy et al. 2001), the Vehicle Routing Problem (Moraga 2002), and the 0-1 MKP (Moraga et al. 2005). DePuy et al. (2005) aimed to develop a simple method to find good solutions to Traveling Salesman Problem (TSP). The Set Covering Problem (SCP) was another optimization problem for which Meta-RaPS performed well (Lan et al. 2007). Rabadi et al. (2006) developed Meta-RaPS for

```
For iteration ≤ I
While (feasible solution is not constructed)
      Find priority value for each feasible activity
      Find best priority value
      If rand() ≤ p% then
               add item with best priority value to solution
      Else create CandidateList from feasible activities with
               priority values ≥ Limit
      Limit = MinimumPriority +
               r% ·(MaximumPriority – MinimumPriority)
      Choose randomly an item from CandidateList and add to solution
End While
Δ = BestConstructedSolution · i%
If ConstructedSolution ≥ Δ then improve
If ImprovedSolution > BestImprovedSolution then
   Assign ImprovedSolution as BestImprovedSolution
End For
Report BestImprovedSolution
```

Fig. 6.1 Meta-RaPS pseudo code

the unrelated parallel machine scheduling problem with machine-dependent and sequence-dependent setup times to minimize the makespan. Hepdogan et al. (2009) applied Meta-RaPS algorithm to the early/tardy single machine scheduling problem with common due date and sequence-dependent setup times. Garcia and Rabadi (2011) developed a new algorithm based on Meta-RaPS for solving the parallel multiple-area spatial scheduling problem with release times. Kaplan and Rabadi used Meta-RaPS to solve the Aerial Refueling Scheduling Problem (2012), a real-world problem that requires high quality solutions in an acceptable time frame.

6.3 Path Relinking (PR)

The PR approach originally stems from strategies of combining decision rules and constraints in the context of integer programming. The basic idea behind PR is to reinterpret the linear combinations of points in the Euclidean space as paths between and beyond solutions in the neighborhood (Talbi 2009). The PR approach generates new solutions by exploring trajectories connecting the *initiating solution* and the *guiding solution*. While following the path from the initiating towards the guiding solution, high-quality solutions are created by selecting moves with "good" attributes contained in the guiding solution. At each iteration, the move with the best objective function value, which also reduces the distance between the two solutions is selected. This is repeated until the distance is equal to 0 at which point the best solution found in the trajectory is returned by the algorithm.

Initial	Guiding	Neighbors		
0 0 1 **1**	1 1 0 **1**	1 0 1 1*	0 1 1 1	0 0 0 1
1 0 1 **1**	**1** 1 0 **1**	1 1 1 1	1 0 0 1*	
1 0 **0 1**	**1** 1 **0 1**	1 1 0 1*		
1 1 0 1	**1 1 0 1**			

Fig. 6.2 Example of the PR process

To follow the PR process, the initial and guiding solutions obtained in the solution process are coded into a binary string. The positions containing the same numbers in the initial and guiding solutions are identified to keep their states and the numbers in the remaining positions are changed in a systematic way to create a neighborhood. The neighbor with the maximum profit is selected to build the path. At each step, the solutions become more similar to the guiding solution and more different from the initial solution. While progressing, the solution found is replaced with the best improved solution only if it is better than the best improved solution found so far. Figure 6.2 summarizes the PR transformation process from the initial to guide solutions for a 4-item 0-1 MKP problem instance. If items 3 and 4 are selected for the initial solution, and items 1, 2, and 4 for the guiding solution, they will then be coded as (0 0 1 1) and (1 1 0 1), respectively. Note that initial and guiding solutions share only one item (item 4) with the same state at the same position. The states of items in the other positions of the initial solution are switched from *selected* (1) to *not selected* (0), or vice versa to obtain the following neighbors: (1 0 1 1), (0 1 1 1), and (0 0 0 1). The best neighbor, i.e., the one with the maximum profit shown with * in Fig. 6.2, is selected as the new initial solution, which is now closer to the guiding solution, having two items at the same position. This process is repeated until the initial and guiding solutions become identical.

PR is different from local search approaches in several ways: the path between initial and guiding solutions is directed by the criterion of incorporating attributes of the guiding solution, not by local attraction. This feature helps PR reach some solutions that would not be found by a "locally myopic" search (Glover et al. 2003). The relinked path may also provide fertile starting points for creating neighborhoods that may include high quality solutions. Glover and Laguna (1997) suggested PR as an approach to integrate intensification and diversification strategies in the context of TS. PR can be used to diversify or intensify the search, depending on the path generation and the choice of the initial and guiding solutions. For each pair of initial and guiding solutions, there exist different alternatives in selecting the starting and the guiding solutions:

- Forward: The worst of both solutions is used as the starting solution.

- Backward: The better of both solutions is used as the starting solution. Since the starting solution's neighborhood is more explored than that of the target solution, the backward strategy is, in general, better than the forward one.
- Backward and forward relinking: Two paths are constructed in parallel, using alternatively both solutions as the starting and the target solutions.
- Mixed relinking: Two paths are constructed in parallel from both solutions but the guiding solution is an intermediate solution at the same distance from both solutions.

PR was originally proposed by Glover (1996) as a way to explore trajectories between elite solutions obtained by TS or Scatter Search (SS), and later Martí et al. (2005) applied PR within GRASP. PR became an attractive approach applied as an intensification strategy to each local optimum obtained after the local search phase (Resende and Ribeiro 2003), as a post-optimization step to all pairs of elite solutions (Deng and Bard 2011; Villegas et al. 2011), or as both intensification and post-optimization strategies (Resende and Werneck 2002) in GRASP. In the literature, GRASP and PR applications are produced by researchers for many optimization problems, such as scheduling (Alvarez-Valdes et al. 2008; Bozejko 2010), max-min diversity problem (Resende et al. 2010), generalized quadratic assignment problem (Mateus et al. 2011), lot sizing problem (Nascimento et al. 2010), and set k-covering problem (Pessoa et al. 2012).

In addition to GRASP, Andrade and Resende (2007) showed that a GRASP with evolutionary PR finds solutions faster than a heuristic based GRASP with PR or a pure GRASP. Based on the adaptive memory and responsive strategy elements of SS and PR, Yin et al. (2010) created a combination of Particle Swarm Optimization (PSO) and SS/PR to produce a Cyber Swarm Algorithm that proves more effective than standard PSO. There are many other successful hybrid applications in which PR is used to add a memory mechanism by integrating it into other algorithms, including TS (Armentano et al. 2011; Nasiri and Kianfar 2012), Variable Neighborhood Search (Wang and Tang 2009), SS (Nasiri and Kianfar 2011; Ranjbar et al., 2009), ACO (Liu and Liu 2011), and memetic algorithms (Jaszkiewicz and Zielniewicz 2009).

PR has also been applied in continuous optimization (Jaeggi et al. 2008), multiobjective combinatorial optimization (Beausoleil et al. 2008), interior point methods (Plateau et al. 2002), and large-scale global optimization (Duarte et al. 2011). Festa and Resende (2011) gave an overview of GRASP and its enhancements including the PR strategy. Ribeiro and Resende (2012) reviewed PR intensification methods for stochastic local search algorithms. Detailed explanation of PR is presented by Glover et al. (2000), and a survey reporting on advanced PR strategies can be found in (Resende and Ribeiro 2005).

```
For iteration ≤  I
Apply Meta-RaPS rules to produce ImprovedSolution and BestImprovedSolution
    Assign ImprovedSolution as InitialSolution
    Assign BestImprovedSolution as GuidingSolution
    While (InitialSolution ≠ GuidingSolution)
        Create CandidateSolutions
        Assign BestCandidateSolution as PathRelinkingSolution
        If PathRelinkingSolution > BestImprovedSolution then
           Assign PathRelinkingSolution as BestImprovedSolution
        Assign PathRelinkingSolution as InitialSolution
    End While
End For
Report BestImprovedSolution
```

Fig. 6.3 Meta-RaPS PR pseudo code

6.4 Designing Meta-RaPS PR Algorithm

The PR in the proposed algorithm will be applied to the pairs of initial and guiding solutions, where the initial solution is the locally optimal solution obtained after the local search in the improvement phase of Meta-RaPS, and the guiding solution is the best solution found so far over all iterations by the proposed algorithm. The Meta-RaPS PR will learn the attributes of the best solution by memorizing it in the solution history, and applying this information to generate other high quality solutions by creating a path between the new initial solution and the best solution. This changes the memoryless Meta-RaPS into a more "intelligent" algorithm, we call Meta-RaPS PR, that can memorize the solution history and learn the "good" attributes to make smart decisions, without affecting any of its main principles.

The PR phase of Meta-RaPS PR is not executed at the first iteration because the best improved solution to serve as a guiding solution is not constituted yet. The modified Meta-RaPS PR pseudo code is shown in Fig. 6.3.

6.5 Performance Evaluation

The proposed Meta-RaPS PR algorithm was implemented in C++. Due to the use of randomness in Meta-RaPS, the proposed algorithm was run 10 times for each instance and the average was taken for all runs. After completing the solution process, the performance of the algorithm was reported in terms of solution quality, or deviation percentage, number of iterations, CPU time, and frequency of reaching optimum or best solutions. The deviation percentages were calculated using Eq. (6.4):

Table 6.1 Parameters of Meta-RaPS PR for 0-1 MKP instances

	Values	
Parameter	Small/medium 0-1 MKP	Large 0-1 MKP
Priority (p)	0.4	0.60
Restriction (r)	0.2	0.65
Improvement (i)	0.1	0.10
Number of iterations (I)	10,000	10,000/5000/1000

$$Deviation\ \% = \frac{f(s^*) - f(s)}{f(s^*)} \times 100 \tag{6.4}$$

where s is the solution found in the current method and s^* is the optimum or best solution found.

To tune the parameters of Meta-RaPS, D-optimal Design of Experiments (DOE) was applied to tune the parameter settings for small/medium 0-1 MKP instances. In tuning its parameters for large 0-1 MKP instances, an adaptive tuning method, which utilizes feedback information obtained during the search to perform a learning process of the parameter combination, was used offline, i.e., before the proposed algorithm actually started (Alabas-Uslub and Dengiz 2011). To begin with, a parameter memory matrix was created for the parameters *priority* ($p\%$) and *restriction* ($r\%$) parameters, containing 9 levels between 0.1 and 0.9 with increments of 0.1 for each parameter, to memorize and learn the effects of different parameter settings during the solution process. The *improvement* parameter ($i\%$) was accepted as 0.1 according to an initial analysis; thus, 81 ($=9 \times 9$) different parameter settings could be attempted in solving the 0-1 MKP instances. These parameter settings were then applied in solving the instances and the best solution values were recorded in the cells representing the parameter settings of $p\%$ and $r\%$ until the parameter memory matrix converged. Finally, the overall average values of $p\%$ and $r\%$ that created the best solutions for each instance were accepted as the best parameter setting as presented in Table 6.1. For the large size 0-1 MKP instances, the parameter *number of iteration* (I) was accepted as 10,000 for instances with 100 items, 5000 for instances with 250 items, and 1000 for instances with 100 items.

To evaluate the performance of the proposed algorithm, it was applied to 55 small/medium size (Petersen 1967; Weingartner and Ness 1967; Shih 1979; Fréville and Plateau 1990), and 270 large size (Chu and Beasley 1998) 0-1 MKP test instances available in the OR-Library (Beasley 1990). For the large instances, the 30 instances in each set were created with tightness ratios (T/R) of 0.25, 0.50, and 0.75 for each group of 10 instances in the set. T/R is defined as the ratio between the constraint's right-hand side value and the sum of the corresponding weights as given in Eq. (6.5).

$$\mathrm{T/R} = \frac{b_i}{\sum_{j=1}^{n} a_{ij}} \in \{0.25, 0.5, 0.75\} \tag{6.5}$$

Resource consumptions a_{ij} are random numbers generated between (0, 1000), and profit values c_j are generated via Eq. (6.6).

$$c_j = \sum_{i=1}^{m} \frac{a_{ij}}{m} + (500r_i) \in \{0.25, 0.5, 0.75\} \tag{6.6}$$

where m is the number of knapsacks, r_i is a random number generated from (0, 1).

6.6 Computational Results

In the original Meta-RaPS, the solutions were constructed by employing a greedy rule called Dynamic Greedy Rule (DGR) as a priority rule in determining the priorities or order of the items to be selected (Moraga et al. 2005). In the proposed Meta-RaPS PR algorithm, the same greedy approach will be followed to generate solutions in the construction phase by repeatedly adding feasible items to the current (partial) solution based on their DGR values until a solution is generated, i.e., until there is no more feasible items to add to a partial solution. In this rule, a penalty factor for each item is calculated according to Eq. (6.7).

$$w_i = \sum_{j=1}^{m} \frac{a_{ij}}{b_j - CW_j}, \quad \text{for } i = 1, \ldots, n \tag{6.7}$$

where a_{ij} is the coefficients of item i in constraint j, b_j is the amount of resource for each constraint j, and CW_j is the amount of resource j consumed by the items (or partial solution) so far. To determine the priority of an item i, its profit c_i is divided by its penalty factor, i.e., c_i/w_i. The item with maximum c_i/w_i has the highest priority in the solution process. Because the penalty factor changes after each iteration in the construction process, the priorities of the items are updated after each item is added to the partial solution.

In the improvement phase of Meta-RaPS PR, two different algorithms are employed: 2-opt and insertion algorithms. In 2-opt algorithm, a randomly selected item in the solution is replaced with another randomly selected item that is not in the solution in a systematic way. In the insertion algorithm, a randomly selected item is inserted before or after another item in the solution and items between the old and new places of inserted item are shifted towards the old place of the inserted item in the same order. Other items remain in their positions.

The deviation percentages for the solutions were calculated not only for the average of best improved solutions (IMean), but also for the best of the best improved solutions found in 10 runs of Meta-RaPS PR (IBest). While IMean shows the mean performance of the algorithm, IBest gives an idea about the limits of the algorithm (see Table 6.2). In terms of the frequency of reaching optimum or best solutions, the number of times the algorithm found the optimum or best

Table 6.2 Meta-RaPS PR results for small/medium 0-1 MKP instances

Deviation%		Number of iteration	Time (s)	Optimum	
IMean	IBest			Frequency	Instances
0.001	0.000	480	47.93	9.76	55/55

solutions in 10 runs is given under the heading of Optimum Frequency. The heading "Optimum Instances" shows the number of instances solved optimally or for which best solutions were found by the algorithm. Finally, the averages and standard deviations for these metrics are calculated. The stopping criteria for the proposed algorithms are to run the algorithms until the number of iteration is met or, to stop whenever the deviation percentage from the optimal or best found solution becomes 0, whichever comes first.

6.6.1 Meta-RaPS PR for Small and Medium 0-1 MKP Instances

Meta-RaPS PR was first applied to small and medium 0-1 MKP instances. As summarized in Table 6.2, Meta-RaPS PR could solve all small and medium instances in the OR-Library, and was able to find optimum or best solutions 9.8 out of 10 runs on average for all instances. The average deviation percentage reached by the proposed algorithm was 0.001 %. Meta-RaPS PR obtained these results in an average of 48 s and 480 iterations, respectively, on an Intel i5 CPU 2.27 GHz PC.

Table 6.3 presents a comparison of Meta-RaPS PR to other approaches in the literature including the original Meta-RaPS DGR for the small/medium 0-1 MKP instances. The ratios in the table indicate the number of instances solved optimally out of the total number of instances attempted. TS methods (Glover and Kochenberger 1996; Hanafi et al. 1996), GA (Chu and Beasley 1998), and Fix + cut-based method (Osorio et al. 2003) generated the best results in the literature. Meta-RaPS PR could create considerably good results in terms of the number of optimal solutions and deviation percentages. Meta-RaPS DGR attempted to solve the same 55 instances in their 56 instance-set, and PETERSEN7 was the only instance for which Meta-RaPS DGR did not find the optimal solution; however, Meta-RaPS PR could solve this instance optimally. Data and solutions for these instances are available at www.SchedulingResearch.com

To reveal the contribution of the PR to Meta-RaPS, the number of optimum or best solutions found in the improvement phase and PR phase are tracked for each instance. To better see the impact of each phase, the instances are put in the order of their size, which is defined here as the product of the number of items by the number of knapsacks. The contribution of both phases in reaching optimal or best solutions in the new Meta-RaPS design is reported. Figures 6.4 and 6.5 show the frequency

Table 6.3 Comparison of Meta-RaPS PR to other algorithms in the literature for small/medium 0-1 MKP instances (adapted from Moraga et al. 2005)

Algorithm	#Optimal solutions	Deviation%
Meta-RaPS PR	55/55	0.001
Meta-RaPS DGR (Moraga et al. 2005)	55/56	0.003
GRASP (Moraga et al. 2005)	52/56	0.023
SMA/TA (Hanafi et al. 1996)	39/54	0.080
AGNES (Fréville and Plateau 1994)	52/54	0.020
Tabu search L + STM (Dammeyer and Voss 1993)	44/57	0.101
Tabu search (Glover and Kochenberger 1996)	57/57	0.000
Tabu search (Løkketangen and Glover 1998)	37/54	0.003
Tabu search IFTS/HFE (Hanafi et al. 1996)	54/54	0.000
Genetic algorithm (Chu and Beasley 1998)	55/55	0.000
Fix + cut based method (Osorio et al. 2003)	55/55	0.000
Simulated annealing PROEXC (Drexl 1988)	23/57	0.239
Simulated annealing (Drexl 1988; Dammeyer and Voss 1993)	31/57	0.328

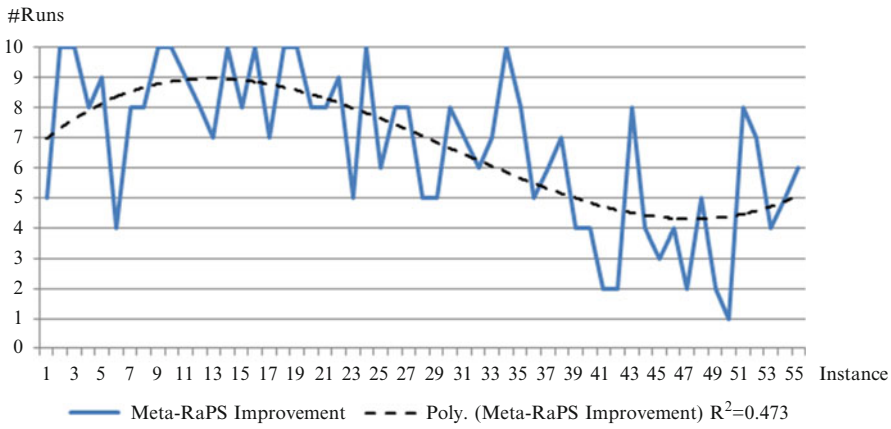


Fig. 6.4 Trend line of best solutions found in 10 runs by improvement phase of Meta-RaPS PR for small/medium instances

of best and optimal solutions reached during the improvement phase in 10 runs per instance.

As can be seen from the trendline, the effectiveness of reaching such solutions declines as the size of the instances increases. In Fig. 6.4, the contribution of the PR phase is measured in the same way and the trendline indicates that the finding optimal and best solutions during the PR phase increases with the increase in the instance size. Especially for the larger instances, the role of PR in the proposed algorithm is clear; its share of number of optimum or best solutions found in 10 runs for each instance increased with the increase in instance size.

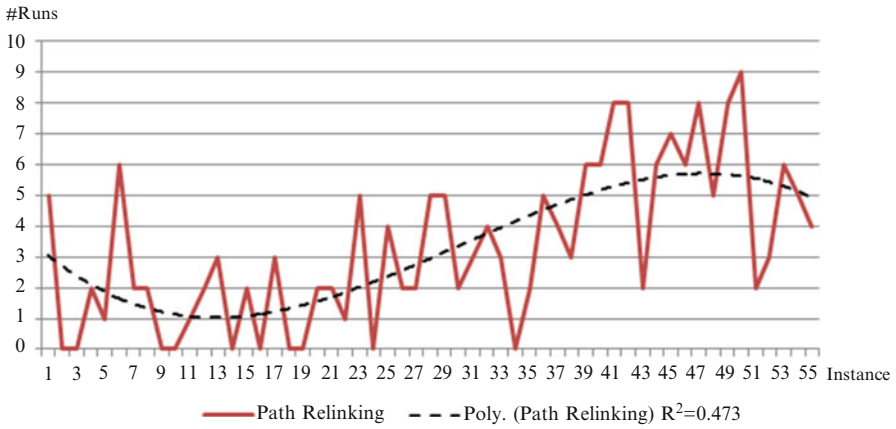


Fig. 6.5 Trend line of best solutions found in 10 runs by improvement phase of Meta-RaPS PR for small/medium instances

Recall that besides the number of iterations parameter (I), there is another stopping criterion, which is when the deviation percentage is equal to 0. For the instances with smaller size, Meta-RaPS could find optimum solutions, and stops the solution process before reaching the PR phase. This was the reason why the PR phase seems to not produce optimal or best solutions for these instances. However, in solving large instances, Meta-RaPS had to reach to the PR phase to obtain better results.

6.6.2 Meta-RaPS PR for Large 0-1 MKP Instances

With the parameters set to the values in Table 6.1, Meta-RaPS PR was run to solve the large size 0-1 MKP instances of 100, 250, and 500 items by 5, 10, and 30 knapsacks, respectively. Table 6.4 shows that the overall average deviation percentage from the optimum or best solution reached by Meta-RaPS PR is 0.276 % in an average of 1674 iterations (Meta-RaPS PR was run 10 times per instance). Its average optimum or best solution was 2.1 in the 30 instance set, and the average optimum or best solution was found 1.5 in 10 runs per instance.

A comparison of the proposed Meta-RaPS PR algorithm to other algorithms in the literature for large 0-1 MKP instances only in terms of deviation percentage is presented in Table 6.5 due to absence of the information about standard deviations for these approaches. Data and solutions for these instances are available at www.SchedulingResearch.com

In addition to comparing the newly introduced Meta-RaPS PR algorithm with the memoryless Meta-RaPS (DGR) of Moraga et al. (2005), it was also compared to the following existing algorithms: a new problem reduction heuristic NR(P) (Hill et al.

Table 6.4 Meta-RaPS PR solutions for large 0-1 MKP instances

Instance set	Deviation% IMean	Number of IBest	Time iterations	Optimum (min)	Frequency	Instances
100 × 5	0.017	0.000	1612	1.39	7.90	9.67
100 × 10	0.182	0.109	2552	4.06	3.83	4.67
100 × 30	0.413	0.252	3193	12.60	0.43	1.00
250 × 5	0.095	0.040	2037	29.38	1.27	2.67
250 × 10	0.224	0.136	2111	47.03	0.17	0.67
250 × 30	0.543	0.007	2061	108.77	0.00	0.00
500 × 5	0.162	0.117	521	45.13	0.00	0.00
500 × 10	0.269	0.192	498	65.23	0.00	0.00
500 × 30	0.576	0.430	478	142.13	0.00	0.00
Average	0.276	0.142	1674	50.64	1.51	2.07

Table 6.5 Comparison of Meta-RaPS PR to other algorithms for large 0-1 MKP instances (adapted from Moraga et al. 2005)

	Meta-RaPS							
Instance set	PR	DGR	NR(P)	HDP-LBC	MMA-8	PECH	GA-CB	GA-HV
100 × 5	0.01	0.60	0.53	0.57	0.80	4.28	0.59	0.72
100 × 10	0.18	1.17	1.10	0.32	1.32	4.58	0.94	1.26
100 × 30	0.41	2.23	1.45	1.81	2.07	1.69	1.69	2.14
250 × 5	0.09	0.17	0.24	0.16	0.43	4.03	0.14	0.36
250 × 10	0.22	0.45	0.48	0.32	0.70	3.31	0.30	0.74
250 × 30	0.54	1.38	0.80	0.77	1.14	2.69	0.68	1.36
500 × 5	0.16	0.09	0.08	0.07	0.34	3.85	0.05	0.34
500 × 10	0.27	0.20	0.19	0.16	0.57	2.92	0.14	0.64
500 × 30	0.57	0.82	0.49	0.42	0.89	2.09	0.35	1.20
Average	0.28	0.79	0.60	0.51	0.92	3.27	0.54	0.97

2012), a surrogate and a dynamic programming method with a limited branch and cut (LBC) improvement phase, HDP-LBC (Boyer et al. 2009), Memetic Algorithm approach, MMA-8 (Özcan and Başaran 2009), a greedy-like heuristic method, Primal Effective Capacity Heuristic (PECH) (Akcay et al. 2007), Genetic Algorithm GA-CB (Chu and Beasley 1998), and Genetic Algorithm GA-HV (Haul and Voss 1997). As can be seen from the table, the proposed Meta-RaPS PR produced better results than all of the other algorithms in the literature. Specifically, redesigning Meta-RaPS from a memoryless algorithm to a more intelligent algorithm clearly demonstrates its enhanced ability to find better solutions across all instances of large MKP problems. It is difficult and not meaningful to make similar comparisons with older studies in terms of CPU due to evolving computer technology.

6.7 Conclusions

Intensification and diversification in metaheuristics are two main strategies to improve the quality of solutions in the search process. In Meta-RaPS, which is a memoryless metaheuristic, intensification and diversification strategies are controlled only by randomness as specified by its parameters. In addition to randomness, Path Relinking (PR) was integrated into Meta-RaPS as a learning algorithm for intensification purposes by learning "good" attributes of the best solutions. Applying PR as an intensification strategy to Meta-RaPS was very effective in creating a better performance. The PR approach also changed the memoryless nature of Meta-RaPS to a more "intelligent" algorithm. The new algorithm presented very good results for a test problem of the Multidimensional Knapsack Problem (MKP) compared to other approaches in the literature.

The PR method used as both an intensification strategy and a learning mechanism in Meta-RaPS was very successful at obtaining high quality solutions without affecting the main principles of Meta-RaPS, and therefore, this approach can be conveniently applied in other metaheuristics algorithms in the future.

References

Akcay Y, Li H, Xu S (2007) Greedy algorithm for the general multidimensional knapsack problem. Ann Oper Res 150(1):17–29

Alabas-Uslub C, Dengiz B (2011) A self-adaptive local search algorithm for the classical vehicle routing problem. Expert Syst Appl 38:8990–8998

Alvarez-Valdes R, Crespo E, Tamarit JM, Villa F (2008) GRASP and path relinking for project scheduling under partially renewable resources. Eur J Oper Res 189:1153–1170

Andrade DV, Resende MGC (2007) GRASP with evolutionary path-relinking. AT&T Labs Research Technical Report TD-6XPTS7

Arcus AL (1966) COMSOAL: a computer method of sequencing operations for assembly lines. Int J Prod Res 4(4):259–277

Arin A, Rabadi G (2012) Memory and learning in metaheuristics. In: Xin-She Y (ed) Artificial intelligence, evolutionary computation and metaheuristics in studies in computational intelligence. Springer, Verlag Berlin Heidelberg, pp 435–476

Armentano VA, Shiguemoto AL, Løkketangen A (2011) Tabu search with path relinking for an integrated production-distribution problem. Comput Oper Res 38(8):1199–1209

Beasley JE (1990) OR-library: distributing test problems by electronic mail. J Oper J Soc 41:170–181. http://people.brunel.ac.uk/~mastjjb/jeb/info.html

Beausoleil RP, Baldoquin G, Montejo RA (2008) Multi-Start and path relinking methods to deal with multiobjective knapsack problems. Ann Oper Res 157:105–133

Boyer V, Elkihel M, El Baz D (2009) Heuristics for the 0-1 multidimensional knapsack problem. Eur J Oper Res 199:658–664

Bozejko W (2010) Parallel path relinking method for the single machine total weighted tardiness problem with sequence-dependent setups. J Intell Manuf 21(6):777–785

Chu PC, Beasley JE (1998) A genetic algorithm for the multidimensional knapsack problem. J Heuristics 4:63–86

Dammeyer F, Voss S (1993) Dynamic tabu list management using the reverse elimination method. Ann Oper Res 41:31–46

Deng Y, Bard JF (2011) A reactive GRASP with path relinking for capacitated clustering. J Heuristics 17:119–152

DePuy GW, Whitehouse GE (2001) A simple and effective heuristic for the resource constrained project scheduling problem. Int J Prod Res 39(14):3275–3287

DePuy GW, Whitehouse GE, Moraga RJ (2001) Meta-RaPS: a simple and efficient approach for solving combinatorial problems. Paper presented at the 29th international conference on computers and industrial engineering, Montreal, 1–3 November, pp 644–649

DePuy GW, Moraga RJ, Whitehouse GE (2005) Meta-RaPS: a simple and effective approach for solving the traveling salesman problem. Transp Res Part E Logistics Transp Rev 41(2):115–130

Dréo J, Aumasson J-P, Tfaili W, Siarry P (2007) Adaptive learning search, a new tool to help comprehending metaheuristics. Int J Artif Intell Tools 16(3):1–23

Drexl A (1988) A simulating annealing approach to multiconstraint zero-one knapsack problem. Computing 40:1–8

Duarte A, Martí R, Gortazar F (2011) Path relinking for large-scale global optimization. Soft Comput 15(11):2257–2273

Festa P, Resende MGC (2011) GRASP: basic components and enhancements. Telecommun Syst 46:253–271

Fréville A, Plateau G (1990) Hard 0-1 multiknapsack test problems for size reduction methods. Invest Operativa 1:251–270

Fréville A, Plateau G (1994) An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. Discrete Appl Math 49:189–212

Garcia C, Rabadi G (2011) A Meta-RaPS algorithm for spatial scheduling with release times. Int J Planning Scheduling 1(1/2):19–31

Garey MR, Johnson DJ (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, San Francisco

Glover F (1996) Tabu search and adaptive memory programing - advances, applications and challenges. In: Barr RS, Helgason RV, Kennington JL (eds) Interfaces in computer science and operations research. Kluwer Academic, New York, pp 1–75

Glover F, Kochenberger G (1996) Critical event tabu search for multidimensional knapsack problems. In: Osman IH, Kelly JP (eds) Meta-heuristics: theory and applications. Kluwer Academics, Dordrecht, pp 407–427

Glover F, Laguna M (1997) Tabu search. Kluwer Academic, Boston

Glover F, Laguna M, Marti R (2000) Fundamentals of scatter search and path relinking. Control Cybern 29(3):653–684

Glover F, Laguna M, Marti R (2003) Scatter search and path linking. In: Glover F, Kochenberger GA (eds) Handbook of metaheuristics. Kluwer Academic, Boston

Hanafi S, Fréville A, El Abdellaqui A (1996) Comparison of heuristics for the 0-1 multidimensional knapsack problem. In: Osman IH, Kelly JP (eds) Meta-heuristics: theory and applications. Kluwer Academics, Dordrecht, pp 449–465

Haul C, Voss S (1997) Using surrogate constraints in genetic algorithms for solving multidimensional knapsack problems. In: Woodruff DL (ed) Advances in computational and stochastic optimization, logic programming, and heuristic search, interfaces in computer science and operation research. Kluwer Academic, New York, pp 235–251

Hepdogan S, Moraga RJ, DePuy GW, Whitehouse GE (2009) A Meta-RaPS for the early/tardy single machine scheduling problem. Int J Prod Res 47(7):1717–1732

Hill RR, Cho YK, Moore JT (2012) Problem reduction heuristic for the 0-1 multidimensional knapsack problem. Comput Oper Res 39:19–26

Jaeggi DM, Parks GT, Kipouros T, Clarkson PJ (2008) The development of a multi-objective tabu search algorithm for continuous optimisation problems. Eur J Oper Res 185(3):1192–1212

Jaszkiewicz A, Zielniewicz P (2009) Pareto memetic algorithm with path relinking for bi-objective traveling salesperson problem. Eur J Oper Res 193(3):885–890

Kaplan S, Rabadi G (2012) Exact and heuristic algorithms for the aerial refueling parallel machine scheduling problem withduedate-to-deadlinewindowandreadytimes. Comput Ind Eng 62(1):276–285

Lan G, DePuy GW, Whitehouse GE (2007) An effective and simple heuristic for the set covering problem. Eur J Oper Res 176:1387–1403
Liu Y-F, Liu S-Y (2011) A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. Appl Soft Comput J. doi:10.1016/j.asoc.2011.10.024
Løkketangen A, Glover F (1998) Solving zero-one mixed integer programming problems using tabu search. Eur J Oper Res 106:624–658
Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. Wiley, Chichester
Martí R, Montes F, El-Fallahi A (2005) Approximating unknown mappings: an experimental evaluation. J Heuristics 11:219–232
Mateus GR, Resende MGC, Silva RMA (2011) GRASP with path-relinking for the generalized quadratic assignment problem. J Heuristics 17(5):527–565
Moraga RJ (2002) Meta-RaPS: an effective solution approach for combinatorial problems. Ph.D. Dissertation, University of Central Florida, Orlando
Moraga RJ, DePuy GW, Whitehouse GE (2005) Meta-RaPS approach for the 0-1 multidimensional knapsack problem. Comput Ind Eng 48:83–96
Nascimento MCV, Resende MGC, Toledo FMB (2010) GRASP heuristic with path-relinking for the multi-plant capacitated lot sizing problem. Eur J Oper Res 200(3):747–754
Nasiri MM, Kianfar F (2011) A hybrid scatter search for the partial job shop scheduling problem. Int J Adv Manuf Technol 52:1031–1038
Nasiri MM, Kianfar F (2012) A guided tabu search/path relinking algorithm for the job shop problem. Int J Adv Manuf Technol 58:1105–1113
Osorio MA, Glover F, Hammer P (2003) Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. Ann Oper Res 117:71–93
Özcan E, Başaran C (2009) A case study of memetic algorithms for constraint optimization. Soft Comput 13:871–882
Pessoa LS, Resende MGC, Ribeiro CC (2012) A hybrid Lagrangean heuristic with grasp and path-relinking for set K-covering. Comput Oper Res. doi:10.1016/j.cor.2011.11.018
Petersen CC (1967) Computational experience with variants of the Balas algorithm applied to the selection of R&D projects. Manag Sci 13(9):736–750
Plateau A, Tachat D, Tolla P (2002) A hybrid search combining interior point methods and metaheuristics for 0-1 programming. Int Trans Oper Res 9:731–746
Rabadi G, Moraga R, Al-Salem A (2006) Heuristics for the unrelated parallel machine scheduling problem with setup times. J Intell Manuf 17:85–97
Ranjbar M, Reyck BD, Kianfar F (2009) A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. Eur J Oper Res 193:35–48
Resende MGC, Ribeiro CC (2003) GRASP with path-relinking for private virtual circuit routing. Networks 41:104–114
Resende MGC, Ribeiro CC (2005) GRASP with path-relinking: recent advances and applications. In: Ibaraki T, Nonobe K, Yagiura M (eds) Metaheuristics: progress as real problem solvers. Springer, Berlin, pp 29–63
Resende MGC, Werneck RF (2002) A GRASP with path-relinking for the P-Median problem. Technical Report, AT&T Labs Research, Florham Park
Resende MGC, Martí R, Gallego M, Duarte A (2010) GRASP and path relinking for the max-min diversity problem. Comput Oper Res 37(3):498–508
Ribeiro CC, Resende MGC (2012) Path-relinking intensification methods for stochastic local search algorithms. J Heuristics 18:193–214
Rochat Y, Taillard E (1995) Probabilistic diversification and intensification in local search for vehicle routing. J Heuristics 1(1):147–167
Shih W (1979) A branch and bound method for the multi-constraint zero-one knapsack problem. J Oper Res Soc 30:369–378
Talbi EG (2009) Metaheuristics: from design to implementation. Wiley, New Jersey
Villegas JG, Prins C, Prodhon C, Medaglia AL, Velasco N (2011) A grasp with evolutionary path relinking for the truck and trailer routing problem. Comput Oper Res 38:1319–1334

Wang X, Tang L (2009) A population-based variable neighborhood search for the single machine total weighted tardiness problem. Comput Oper Res 36:2105–2110
Weingartner HM, Ness DN (1967) Methods for the solution of the multi-dimensional 0/1 knapsack problem. Oper Res 15(1):83–103
Yin P-Y, Glover F, Laguna M, Zhu J-X (2010) Cyber swarm algorithms – improving particle swarm optimization using adaptive memory strategies. Eur J Oper Res 201:377–389

Chapter 7
Meta-RaPS for a Bi-objective Unrelated Parallel Machine Scheduling Problem

Nixon Dcoutho and Reinaldo Moraga

Abstract This chapter discusses the capability and effectiveness of a Meta-heuristic for Randomized Priority Search to solve multi-objective problems. The multi-objective problem of unrelated parallel machine scheduling is considered in the chapter. The two objectives to minimize are total weighted tardiness and total weighted completion time. An existing construction rule in the literature named Apparent Tardiness Cost-bi heuristic is used as the basis for the meta-heuristic construction phase to generate non-dominated solutions. The computational results obtained are promising when results of the meta-heuristic approach proposed are compared with those of the original construction rule. This chapter illustrates that the meta-heuristic approach proposed is effective and flexible enough to generate Pareto-frontiers in order to solve multi-objective scheduling problems by modifying a simple existing heuristic found in the literature.

Keywords Unrelated parallel machine • Bi-objective • Meta-heuristics • Meta-RaPS • Pareto-frontiers

7.1 Introduction

Scheduling refers to a decision-making process of allocating resources to a set of jobs in order to achieve a particular objective. Scheduling can be applied in many industries such as manufacturing, transportation, services, and healthcare. In general, manufacturing scheduling environments occur in a single stage (single or parallel machines) or multiple stage systems (flow shops, job shops, and open shops). In the former systems, only one unique operation is performed on the job before it departs from the system. In the latter systems, jobs visit a number of m machines where operations are performed.

N. Dcoutho • R. Moraga (✉)
Department of Industrial and Systems Engineering, Northern Illinois University, 590 Garden Road, Engineering Building 232, DeKalb, IL 60115, USA
e-mail: rmoraga@niu.edu

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_7

In the case of parallel machines, they can be classified into three types: identical, uniform, or unrelated machines. The scheduling problem of unrelated parallel machine with one single objective is a challenging case (Karp 1972). Nowadays, manufacture scheduling needs to address more than one objective simultaneously due to global competition and other factors. Examples of such objectives include tardiness, earliness, makespan, and the number of tardy jobs among others. Scheduling problems with more than one objective function are referred to in literature as multi-objective scheduling problems, which is the process of finding a schedule that systematically and simultaneously improves all the addressed objective functions.

This book chapter deals with the unrelated parallel machine scheduling problem with minimization of total weighted tardiness ($\Sigma w_j T_j$) and total weighted completion time ($\Sigma w_j C_j$). The same problem with minimization of makespan ($C_{\max}$) is considered NP-hard by Karp (1972), so its multi-objective version is assumed to be NP-hard.

In order to solve the problem of unrelated parallel machine, Meta-heuristic for Randomized Priority Search (Meta-RaPS) will be used. This chapter shows the effectiveness and feasibility of using Meta-RaPS to solve the multi-objective scheduling problem by modifying an existing heuristic found in the literature. The attempt is to show that by using only its construction phase, Meta-RaPS outperforms the existing heuristic.

The rest of this chapter is organized as follows: Sect. 7.2 discusses the existing researches on unrelated parallel machines with multi-objectives. Section 7.3 presents a detailed description and a mathematical model of the proposed problem. This section also contains the explanation for the construction mechanism that will be used. Section 7.4 provides an overview of Meta-RaPS and describes its application. Results and Pareto-frontiers are shown in Sect. 7.5. Finally, Sect. 7.6 summarizes the book chapter and discusses future research opportunities.

7.2 Literature Review

Based on a survey conducted by Lei (2009) multi-objective scheduling problems on unrelated parallel machines is not a well-explored area. The same remark is obtained by a literature review conducted by Kayvanfar et al. (2014a). In general, methods for solving multi-objective unrelated parallel machine scheduling problems can be broadly classified into three groups: priori methods, posteriori methods, and interactive methods. Interactive methods search for feasible solutions by interacting with the decision maker preferences; however, such methods become complex for large problems as more interaction time with the decision maker is required, leading to higher computational efforts. A priori method enables decision maker to set objectives priorities before the search process starts. Posteriori methods generate all non-dominated solutions and then the decision makers select one from the pool according to their preferences. This approach seems to have more practical benefits in real-world applications as it provides more flexibility to the decision-making

process. To the best of our knowledge there are no approaches in the literature that uses a priori method to solve the proposed problem in this book chapter; but Eren (2009) proposes a priori method to minimize a single scalar objective function of total tardiness and total earliness.

Researchers used heuristic methods to solve multi-objective scheduling problems on unrelated parallel machines. For example, Kayvanfar et al. (2014b) address the problem of minimizing total tardiness and total earliness with a restriction on processing times. The authors propose a heuristic known as a Parallel Net Benefit Compression-Net Benefit Expansion (PNBC–NBE) as well as a genetic algorithm (GA) based meta-heuristics. The results of GA based meta-heuristic show promising results when compared to PNBC–NBE heuristics. Lin et al. (2013) use a heuristic known as Apparent Tardiness Cost-bi (ATC-bi) to solve multi-objective unrelated parallel machine scheduling with the goal of minimizing total weighted tardiness and total weighted completion time and the results outperform a GA.

Literature is relatively more abundant with articles reporting the use of meta-heuristics to solve multi-objective unrelated parallel machine scheduling problems due to their easy application and ability to generate solutions under polynomial time (Lei 2009). Moghaddam et al. (2009) use GA to minimize the total completion time and the number of tardy jobs on unrelated parallel machines. Chyu and Chang (2010) address the multi-objective unrelated parallel machine scheduling problem using simulated annealing (SA) to minimize total tardiness and total flowtime. Torabi et al. (2013) develop a Multi-Objective Particle Swamp Optimization algorithm in order to minimize the makespan and the total tardiness with restriction on processing time and sequence dependent setup times. Nogueira et al. (2014) use a Greedy Randomized Adaptive Search Procedure (GRASP) to solve multi-objective unrelated parallel machine scheduling with the objectives of minimizing total tardiness and total earliness.

Except for GRASP, most meta-heuristics reported in the literature use local search mechanisms to solve problems including the multi-objective unrelated parallel machine scheduling problems. In such meta-heuristics there is no balance between the computation spent searching for improved solutions (i.e., local search) versus the time spent constructing solutions. Most meta-heuristics spend most of their time on local search at the expense of the use of construction techniques.

This book chapter presents a Meta-heuristic for Randomized Priority Search (Meta-RaPS) to solve the unrelated parallel machine problem with minimization of total weighted tardiness and total weighted completion time. Meta-RaPS can be defined as a generic, high level strategy used to modify construction heuristics based on the insertion of randomness (DePuy et al. 2005). Meta-RaPS seeks to strike a balance between the usage of the construction phase and the improvement phase. It is a modified version of COMSOAL (Computer Method of Sequencing Operations for Assembly Lines) which is a computer heuristic approach to solve assembly line balancing problems proposed by Arcus (1966). Meta-RaPS can be seen as a more general case than GRASP (DePuy et al. 2005) and has been successfully used to solve a traveling salesman problem by DePuy et al. (2005), the unrelated parallel machines with makespan with sequence dependent setup times by Rabadi

et al. (2006), the early/tardy single machine scheduling problem by Hepdogan et al. (2009), and the spatial scheduling problem with release times by Garcia and Rabadi (2011).

Some attempts are reported in the literature to solve multi-objective problems using Meta-RaPS. For example, Guo (2013) uses Meta-RaPS to solve multi-objective flexible flow shop scheduling problem with the goal of minimizing total tardiness and makespan. Wang (2007) addresses the problem of single machine scheduling with multi-objectives. However, as of now there is no reported work which uses Meta-RaPS to solve the problem proposed in this chapter.

Therefore, the idea of this current application is to embed the ATC-bi construction rule into Meta-RaPS framework in order to generate good quality solutions for the problem proposed. Pareto-frontier solutions are generated with Meta-RaPS and the results are compared with ATC-bi heuristic. It is important to note that to make a fair comparison, only the construction phase of Meta-RaPS will be compared to ATC-bi without the improvement phase.

7.3 Unrelated Parallel Machine Scheduling Problem with Multi-Objectives

The unrelated parallel machine scheduling problem with minimization of total weighted tardiness and total weighted completion time, $R_M||\sum_{j=1}^{N} w_j T_j, \sum_{j=1}^{N} w_j C_j$, may be described as a set of N independent jobs to be processed without preemption on any of the M unrelated parallel machines. Each machine m is capable of processing only one job at a time. It is assumed that all jobs are available at time zero. Job j has a processing time on machine m represented as p_{jk}, a due date d_j, a completion time C_j, and a weight w_j. In addition, a weighted tardiness penalty is incurred for each time unit of tardiness T_j, given by $T_j = \max(0, C_j - d_j)$. $\sum_{j=1}^{N} w_j T_j$ is the total weighted tardiness and $\sum_{j=1}^{N} w_j C_j$ is the total weighted completion time of all the n independent jobs. Hence the problem can be formally stated as: find a schedule S that minimizes the objective functions $f_1(S) = \sum_{j=1}^{N} w_j T_j$ and $f_2(S) = \sum_{j=1}^{N} w_j C_j$.

7.3.1 Linear Integer Programming Model

A mathematical model for the proposed problem in this chapter is given below which is similar to the model shown in Kayvanfar et al. (2014a).

Input Parameters

K Machine Index, $m = 1, \ldots, M$
j Job Index, $j = 1, \ldots, N$
M Total number of machines used
N Total number of jobs to be processed
p_{jk} Processing time of job j on machine k
d_j Due date of job j
w_j Weight assigned to job j

Decision Variables

$C_{j,k}$ Completion time of job j on machine k
T_j Tardiness of job j
$x_{i,j,k}$ 1, if job i precedes job j on machine k; 0, otherwise

Mathematical Formulation

$$\text{Minimize} \left(\sum_{m=1}^{M} \sum_{j=1}^{N} w_j C_j, \sum_{m=1}^{M} \sum_{j=1}^{N} w_j T_j \right)$$

Subject to

$$\sum_{k=1}^{M} \sum_{\substack{i=0 \\ i \neq j}}^{N} x_{i,j,k} = 1 \quad \forall\, j = 1, \ldots, N \tag{7.1}$$

$$\sum_{j=1}^{N} x_{0,j,k} \leq 1 \quad \forall\, k = 1, \ldots, M \tag{7.2}$$

$$\sum_{\substack{i=0 \\ i \neq h}}^{N} x_{i,h,k} - \sum_{\substack{j=0 \\ h \neq j}}^{N} x_{h,j,k} = 0 \quad \forall\, h = 1, \ldots, N; \quad k = 1, \ldots, M \tag{7.3}$$

$$C_{0,k} = 0 \quad \forall\, k = 1, \ldots, M \tag{7.4}$$

$$C_{j,k} - P_{j,k} + U\left(1 - x_{i,j,k}\right) \geq C_{i,k} \quad \forall\, i = 1, \ldots, N; j = 1, \ldots, N; k = 1, \ldots, M \tag{7.5}$$

$$T_j \geq C_{j,k} - d_j \quad \forall\, j = 1, \ldots, N; k = 1, \ldots, M \tag{7.6}$$

$$T_j, C_{j,k} \geq 0 \quad \forall\, j = 1, \ldots, N; k = 1, \ldots, M \tag{7.7}$$

$$x_{i,j,k} \in \{0, 1\} \quad \forall\, i = 1, \ldots, N; j = 1, \ldots, N; k = 1, \ldots, M \tag{7.8}$$

where U is a large positive number.

The objective functions to minimize are total weighted completion and total weighted tardiness. Constraint (7.1) ensures that every job j is assigned to only one machine. Constraint (7.2) limits the dummy job 0 to have a maximum of one successor on each machine. Constraint (7.3) ensures that every job h has exactly one successor. Constraint (7.4) sets the value zero for the completion time of dummy job 0. Constraint (7.5) ensures that the completion time for a job j is greater than its preceding job completion time. Constraint (7.6) restricts the minimum possible value of tardiness for each job j. Constraint (7.7) is the non-negativity constraint. Constraint (7.8) restricts the possible value for $x_{i,j,k}$ to binary variables.

7.3.2 Construction Heuristic

As stated earlier, Lin et al. (2013) developed ATC-bi heuristic to solve the problem with the objective of minimizing the total weighted tardiness and total weighted completion time. ATC-bi heuristics uses Apparent Tardiness Cost (ATC), which is a composite dispatching rule that solves the problem of single machine scheduling with total weighted tardiness. ATC is a combination of weighted shortest processing time first, earliest due date first and minimum slack first. ATC-bi heuristic modifies ATC by adding a machine selection criterion. The ATC-bi heuristic is explained as follows.

S: current schedule
f(*S*): *current objective function value*
J^a*: set of unscheduled jobs*
j^**: job with maximum* I_{i^*j} index value
d_j*: due date of job j*
p_{ij}*: processing time of job j on machine i*
t_i*: load on machine i*
w_j*: weight of job j*
M: Number of machines

1: Set $S = \emptyset$, $J^a = \{1, \ldots, n\}$ and $M = \{1, \ldots, m\}$
2: Determine the first available machine i^*, such that $t_{i^*} = \min_{1 \leq i \leq m} t_i$
3: Calculate I_{i^*j} for each job j, where

$$I_{i*j} = \frac{w_j}{p_{i^*j}} e^{\left(\frac{\max\left(d_j - p_{i^*j} - t_{i^*},0\right)}{k\,\overline{p}_{i^*}}\right)}$$

4: Select the job j^* such that $I_{i^*j^*} = \max_{j^* \in I_{i^*j}} \boldsymbol{I_{i^*j}}$.

5: For job j^*, find machine i^{**} such that $p_{i^{**}j} = \min\left\{p_{ij^*} \middle| t_i + p_{ij^*} - d_{j^*} \leq 0,\ 1 \leq i \leq m\right\}$.
If i^{**} does not exist, then calculate i^{**} as $i^{**} = argmin_{1 \leq i \leq m} \left(\boldsymbol{t_i} + \boldsymbol{p_{ij^*}} - \boldsymbol{d_{j^*}}\right)$

6: Assign j^* to machine i^{**}. Update S and delete j^* from J^a

7: If $J^a = \emptyset$, then STOP, else report S and $f(S)$; and go to step 2

Step 2 selects the machine with minimum work load. In step 3 machine i^* with minimum load is selected and index value I_{i^*j} for each job j is calculated based on the processing time of job j on machine i^*, where $\overline{p}_{i^*}$ is the average of the processing times of the remaining jobs of machine i^*. In step 4 the job j^* with maximum index value I_{i^*j} is selected. In step 5, the job j^* is scheduled on to machine i^{**} which is the fastest machine capable of completing the job j^* thereby making it less tardy. The heuristic stops if all the jobs are assigned, else it goes back to step 2. Further discussion and detailed steps of ATC-bi could be found in Lin et al. (2013).

7.4 Meta-RaPS Framework

The execution of Meta-RaPS is controlled by using four parameters: the number of iterations (I), the priority percentage (p), the restriction percentage (r), and the improvement percentage (i). The number of constructed feasible solutions is determined by the number of iterations. In general, a construction heuristic builds a solution by systematically adding feasible items to the current solution. The item with the best priority value is added to the current solution $p\%$ of the time. With both p and r parameters, Meta-RaPS modifies the way a general construction heuristic chooses the next item to add to the solution by occasionally choosing a job that does not have the best priority value; instead, it chooses a job from a candidate list controlled by r. In addition, a solution improvement algorithm can be included in Meta-RaPS by using the i parameter. Once a solution has been constructed, Meta-RaPS may proceed to improve it further through neighborhood search algorithms. The improvement heuristic is performed if the construction phase's solution value is below certain threshold δ, determined by I and the range between both best and worst unimproved solution values found so far. Finally, the best solution from all iterations is reported ($S^*$and $f(S^*)$).

In general, Meta-RaPS can be summarized using the following pseudo-code, assuming a minimization problem:

I: number of iterations
p: priority percentage
r: restriction percentage

i: improvement percentage
S: current schedule
f(*S*): *current objective function*
*S**: *best schedule*
f(*S**): *best objective function value*
B: best constructed solution so far
W: worst constructed solution so far
δ:threshold on the objective function to accept current solution for improvement

1: Set *I, p, r, i,* $S^* = \emptyset$, and $f(S^*) = \text{INF}$; $B = \text{inf}$; $W = \text{-inf}$;
2: Repeat *I* times
3: $S = \text{construction}\ (p, r)$;
4: Update (*B, W*);
5: If $f(S) \leq \delta$ (*i, B, W)*
6: $S = \text{local\_search}\ (S)$;
7: End
8: Update $(S^*, f(S^*))$;
9: End

7.4.1 Meta-RaPS Construction Phase

In this application, only the construction phase of Meta-RaPS is used to solve the proposed problem. Since good constructed solutions lead to good improved solutions, it makes sense to design a good Meta-RaPS construction phase that could eventually eliminate the improvement phase.

Meta-RaPS increases the probability of generating better results by incorporating randomness into a dispatching rule appropriate for the problem at hand. ATC-bi heuristic is fitted into Meta-RaPS framework as shown below.

1: Set $S = \emptyset$, $J^a = \{1, \ldots, n\}$ and $M = \{1, \ldots, m\}$
2: Generate a random number RN - u [0, 1]
3: Determine the first available machine i^*, such that $t_{i^*} = \min_{1 \leq i \leq m} \boldsymbol{t_i}$
4: Calculate I_{i^*j} for each job $j \in J^a$, where

$$I_{i*j} = \frac{w_j}{p_{i^*j}} e^{\left(\frac{\max\left(d_j - p_{i^*j} - t_{i^*}, 0\right)}{k\,\overline{p}_{i^*}}\right)}$$

5: If RN $\leq p/100$, then go to step 6 else go to step 8
6: Select the job j^* such that $I_{i^*j^*} = max_{j* \in I_{i*j}}\ \boldsymbol{I_{i*j}}$.
7: For job j^*, find machine i^{**} such that $p_{i^{**}j} = \min\left\{p_{ij^*} \middle| t_i + p_{ij^*} - d_{j^*} \leq 0,\ 1 \leq i \leq m\right\}$. If i^{**}, does not exist, then calculate i^{**} as $i^{**} = argmin_{1 \leq i \leq m}\ (\boldsymbol{t_i} + \boldsymbol{p_{ij^*}} - \boldsymbol{d_{j^*}})$, go to step 14

8: Calculate a threshold $L = h - \left[(h-l) * \frac{r}{100}\right]$, here $h = max_{j \in I_{i*j}}\ \boldsymbol{I_{i*j}}$ and $l = min_{j \in I_{i*j}}\ \boldsymbol{I_{i*j}}$
9: Form a candidate list CL such that $CL = \left\{\forall\, j \in J^a \middle| I_{i*j} \geq L\right\}$
10: Select j^* randomly from CL
11: Calculate a threshold $WL = v + \left[(u-v) * \frac{r}{100}\right]$, where $v = argmin_{1 \leq i \leq m}$ $(\boldsymbol{t_i} + \boldsymbol{p_{ij^*}} - \boldsymbol{d_{j^*}})$ and $u = argmax_{1 \leq i \leq m}$ $(\boldsymbol{t_i} + \boldsymbol{p_{ij^*}} - \boldsymbol{d_{j^*}})$
12: Form a candidate list WCL such that $WCL = \left\{\forall\, m \in M \middle| \left(t_i + p_{ij^*} - d_{j^*}\right) \leq K\right\}$
13: Select i^{**} randomly from WCL
14: Assign j^* to machine i^{**}. Update S and delete j^* from J^a
15: If $J^a = \emptyset$, then STOP, else report S and $f(S)$; and go to step 2

Meta-RaPS modifies ATC-bi heuristic at step 5 by using parameter p. In step 5 if the random number is less than p then Meta-RaPS performs ATC-bi heuristic as it is, which is steps 6 and 7. Otherwise, Meta-RaPS uses a modified ATC-bi heuristic in steps 8–13. In step 8 a limit L is calculated based on the I_{i*j} index. In step 9 a candidate list CL is formed with jobs whose index I_{i*j} is less than the limit L. In step 10 job j^* is selected randomly from the CL. Similarly, in steps 11 and 12 limit WL is calculated based on machine load and randomly a machine m is selected. If the tardiness of job j^* on machine m is less than WL, then the job j^* is assigned to the machine i^{**}. Meta-RaPS modifies ATC-bi by randomizing it to aid the algorithm escape local minima. The results and comparison between ATC-bi heuristic and ATC-bi Meta-RaPS are discussed in the next section.

7.5 Results

ATC-bi Meta-RaPS was used to solve both small and large problem sets. The set of small problems consists of ten instances of four machines and 20 jobs, whereas large problem set consists of ten instances of 10 machines and 100 jobs[1]. Processing time p_{ij} was generated using uniform distribution [1,100], Weight w_j using uniform distribution [1, 10] and Due Date d_j using Uniform distribution $[P(1-T-R/2), P(1-T+R/2)]$, where $P = \sum_{i=1}^{m}\sum_{j=1}^{n}\left(p_{ij}/m\right)$; Relative Range of Due date, $R = 0.4$; Tardiness Factor, $T = 0.8$. More details for data generation could be found in Lin et al. (2013).

The algorithm was coded in MATLAB, and experiments were run on an Intel i7 personal computer. Results from ATC-bi Meta-RaPS were compared with those of ATC-bi heuristic, as the samples shown in Figs. 7.1 and 7.2. The scaling parameter K (intrinsic to ATC-bi) was set to values (0.1, 0.5, 1, 2). All the Meta-RaPS experiments were executed at parameter values $p = 0.5$, $r = 0.5$, $I = 1000$ and $i = 0$

[1]Data sets and its solutions are available at www.SchedulingResearch.com.

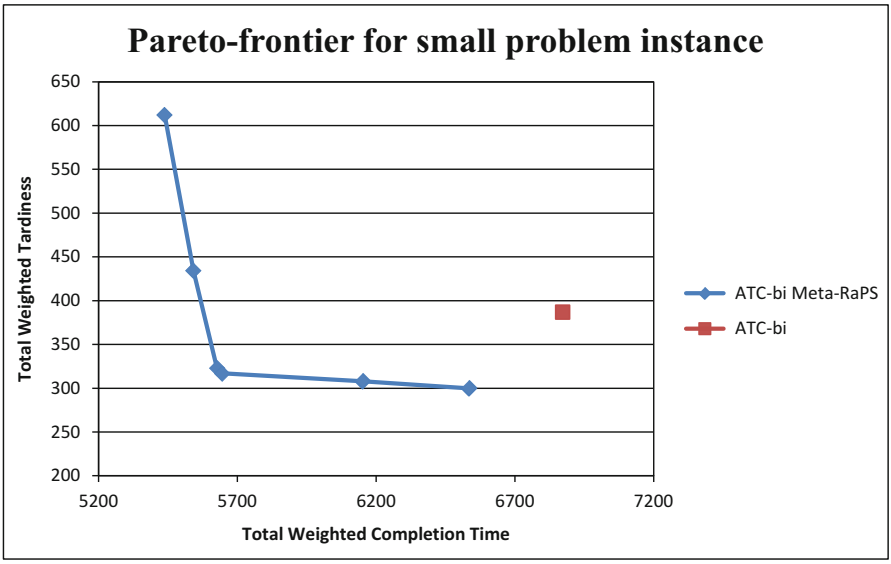


Fig. 7.1 Pareto-frontier results for instance 8 of the small problems

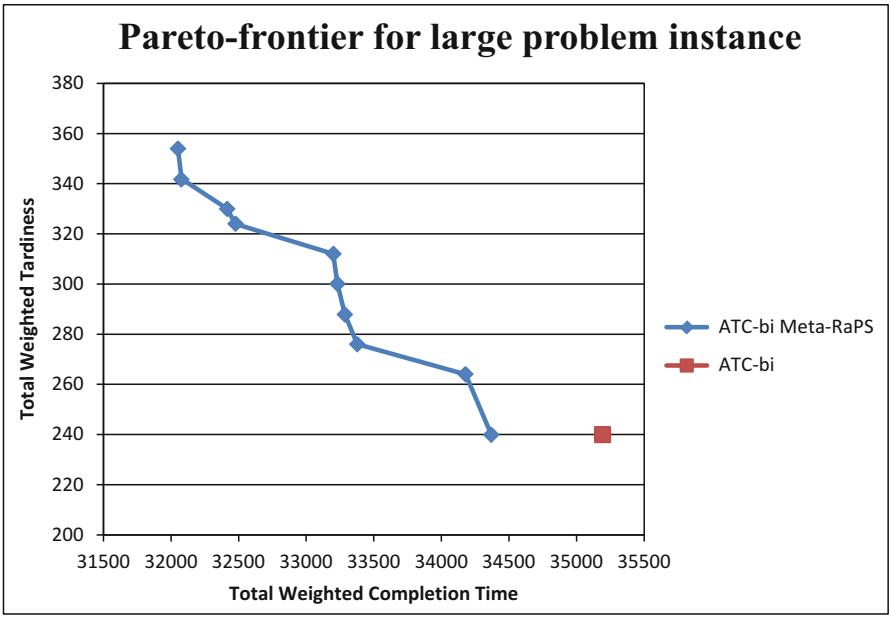


Fig. 7.2 Pareto-frontier results for instance 2 of the large problems

for each K value. ATC-bi heuristics generates only one solution corresponding to each K value as it is a greedy heuristic whereas ATC-bi Meta-RaPS is capable of generating one or more non-dominated solutions as it incorporates randomness into the greedy heuristic.

ATC-bi heuristic was run for each of the four K values thereby generated four solutions and only the non-dominated solutions from this set are shown in results (Tables 7.1 and 7.2). But for a K value, ATC-bi Meta-RaPS was run for 250 iterations and all the 250 solutions are stored in a global variable. Since four K values (0.1, 0.5, 1, 2) are used for this experiment there will be a total of 1000 iterations of ATC-bi Meta-RaPS and all the results are stored in the global variable.

Table 7.1 Non-dominated solutions for $4m20n$ set (Total Weighted Tardiness, Total Weighted Completion Time)

Instance	ATC-bi	ATC-bi Meta-RaPS
1	[108, 6107]	[108, 5872] [111, 5867] [117, 5775] [137, 5669]
2	[0, 5133]	[0, 5028] [60, 5024] [75, 4658]
3	[200, 4955] [147, 6908] [173, 6199]	[147, 6750] [153, 6207] [173, 6130] [180, 4931] [200, 4414]
4	[24, 5377]	[24, 5161]
5	[267, 5903] [243, 6370]	[213, 5762] [222, 5741] [243, 5347] [267, 5160] [330, 4919] [344, 4897]
6	[109, 4730]	[109, 4638]
7	[0, 7320]	[0, 6817]
8	[387, 6872]	[300, 6533] [308, 6152] [317, 5646] [323, 5627] [434, 5542] [612, 5438]
9	[0, 3356]	[0, 3153]
10	[28, 4730]	[28, 4289] [52, 4274]

Table 7.2 Ten non-dominated solutions for $10m100n$ set (Total Weighted Tardiness, Total Weighted Completion Time)

Instance	ATC-bi	ATC-bi Meta-RaPS
1	[8, 32123]	[0, 29772]
2	[240, 35193]	[240, 34368] [264, 34179] [276, 33377] [288, 33286] [300, 33231]
		[312, 33200] [324, 32476] [330, 32415] [342, 32077] [354, 32051]
3	[24, 25578]	[24, 24816]
4	[0, 29062]	[0, 27832]
5	[0, 38121]	[0,36162]
6	[0, 29883]	[0,28683]
7	[84, 27105]	[84, 26247] [90, 25966]
8	[161, 31408]	[77,30506] [161, 30428]
9	[0, 33076]	[0, 32172]
10	[32, 27349]	[32, 26191]

Finally, only the non-dominated solutions are selected from the global variable. Results for small problem instances are shown in Table 7.1 and for large problem instances are shown in Table 7.2.

7.6 Conclusion and Future Research

In this chapter Meta-RaPS construction phase was implemented to solve the unrelated parallel machine scheduling problem with minimization of both total weighted tardiness and total weighted completion time. Meta-RaPS algorithm uses ATC-bi heuristic as its underlying mechanism to construct solutions. Meta-RaPS displays high flexibility by using only a construction phase to generate very good non-dominated solutions. Meta-RaPS is easy to implement as it modifies another heuristic in a simple way to generate better results. Based on our experiments, a good dispatching or composite dispatching rule with Meta-RaPS algorithm can provide high quality solutions for the multi-objective unrelated parallel machine scheduling problem.

In future research, Meta-RaPS can be enhanced by incorporating memory and learning mechanisms in the construction phase, which may greatly enhance its performance by diversifying the region of good feasible solutions. Additionally, the improvement phase (local search) can easily be implemented in Meta-RaPS for this problem and may be compared with other heuristics and meta-heuristics in future studies.

References

Arcus A (1966) A computer method of sequencing operations for assembly lines. Int J Prod Res 4:259–277

DePuy GW, Moraga RJ, Whitehouse GE (2005) Meta-RaPS: a simple and effective approach for solving the traveling salesman problem. Transp Res Part E Logistics Transp Rev 41(2):115–130

Eren T (2009) A bicriteria parallel machine scheduling with a learning effect. Appl Math Model 33:1141–1150

Garcia C, Rabadi G (2011) A Meta-RaPS algorithm for spatial scheduling with release times. Int J Planning Scheduling 1(1/2):19–31

Guo J (2013) A meta-heuristic algorithm to generate Pareto frontiers for a simulation-based flexible flow shop scheduling problem. Master Thesis, Northern Illinois University. Retrieved from ProQuest Database. (Publication No. 1552367)

Hepdogan S, Moraga R, DePuy GW, Whitehouse GE (2009) A Meta-RaPS for the early/tardy single machine scheduling problem. Int J Prod Res 47(7):1717–1732

Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Complexity of computer computations. Plenum Press, New York, pp 85–103

Kayvanfar V, Aalaei A, Hosseininia M, Rajabi M (2014a) Unrelated parallel machine scheduling problem with sequence dependent setup times. In: International conference on industrial engineering and operations management, Bali, pp 7–9

Kayvanfar V, Komaki GM, Aalaei A, Zandieh M (2014b) Minimizing total tardiness and earliness on unrelated parallel machines with controllable processing times. Comput Oper Res 41:31–43

Lei D (2009) Multi-objective production scheduling: a survey. Int J Adv Manuf Technol 43(9–10):926–938

Lin YK, Fowler JW, Pfund ME (2013) Multiple-objective heuristics for scheduling unrelated parallel machines. Eur J Oper Res 227(2):239–253

Moghaddam RT, Taheri F, Bazzazi M, Izadi M, Sassani F (2009) Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. Comput Oper Res 36(12):3224–3230

Nogueira JC, Arroyo JC, Villadiego HM, Goncalves LB (2014) Hybrid GRASP heuristics to solve an unrelated parallel machine scheduling problem with earliness and tardiness penalties. Theor Comput Sci 302:53–72

Rabadi G, Moraga RJ, Salem AA (2006) Heuristics for the unrelated parallel machine scheduling problem with setup times. J Intell Manuf 17(1):85–97

Torabi S, Sahebjamnia N, Mansouri S, Bajestani MA (2013) A particle swarm optimization for a fuzzy multi-objective unrelated parallel machines scheduling problem. Appl Soft Comput 13(12):4750–4762

Wang X (2007) Adaptation of the Meta-RaPS meta-heuristic to a multi-objective environment. Master's thesis, Retrieved from WorldCat Database. (OCLC No. 429700897)

Chapter 8
Heuristics and Meta-heuristics for Runway Scheduling Problems

Farbod Farhadi

Abstract This chapter addresses the state-of-the-art heuristic and meta-heuristic approaches for solving aircraft runway scheduling problem under variety of settings. Runway scheduling has been one of the emerging challenges in air traffic control as the congestion figures continue to rise. From a modeling point of view, mixed-integer programming formulations for single and multiple dependent and independent runways are presented. A set partitioning reformulation of the problem is demonstrated which suggests development of a column generation scheme. From a solution methodology viewpoint, generic heuristic algorithms, optimization-based approaches, and a dynamic programming scheme within the column generation algorithm are presented. Common meta-heuristic approaches that model variant problem settings under static and dynamic environments are discussed.

Keywords Runway scheduling • Mixed-integer programming • Dynamic programming • Optimization-based heuristics • Meta-heuristics

8.1 Introduction and Motivation

Managing challenges of aircraft operations at airports continue to escalate as the air traffic continues to grow at a rapid pace. It has been anticipated that the number of passengers traveling on domestic and international flights will double by year 2025 (Airport Council International). Despite the growing air traffic demands for capacity expansions and infrastructure investments, new strategies and managerial initiatives that best avail of existing infrastructure will continue to be part of the current and future challenges.

In air transport system all airborne and ground-based operations converge at runways, and therefore constitute a key bottleneck for traffic control. Extensive study has been contributed to aircraft runway scheduling problem (ASP). The

F. Farhadi (✉)
Roger Williams University, One Old Ferry Rd, Bristol, RI 02809, USA
e-mail: ffarhadi@rwu.edu

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_8

main parameter in scheduling aircraft on the runways is the sequence dependent minimum separation times, given the operation type (arrival/departure) and the aircraft size. ASP is classified as NP-hard and the computational effort is likely to grow exponentially by the size of the problem.

Various cases of ASP are explored and substantial number of solution methods are examined in the literature. Different settings are considered including scheduling of landings, take offs, or mixed operations; on a single, or multiple dependent, or independent runways; in a static, or a dynamic environment. In this chapter we introduce the state-of-the-art methodologies in the literature for formulating ASP, and the heuristic and meta-heuristic techniques designed to solve ASP. Pure optimization and variations of branching algorithms, as well as dynamic programming approaches are not within the confines of this paper and are subject to another study.

The remainder of this chapter is organized as follows. In Sect. 8.2 we formally introduce the ASP. Section 8.3 summarizes the existing mixed-integer programming (MIP) mathematical formulations for different settings of ASP. Heuristic approaches including generic heuristic, optimization-based heuristics, and a column generation scheme are presented in Sect. 8.4. Meta-heuristic algorithms designed to solve static and dynamic cases of ASP are introduced and briefly described in Sect. 8.5. In conclusion, Sect. 8.6 summarizes the final remarks and possible future directions for research.

8.2 Problem Description

Consider the set J consisting of n aircraft that are ready to be scheduled for landing or take off on an available runway. We define set R as the set of m runways. Each aircraft has an earliest start time $\{e_i | i \in J\}$ and a latest start time $\{l_i | i \in J\}$ which determines the earliest and latest allowable time to schedule the operating time of the aircraft on the runway. The variable t_i is to determine the operation time of aircraft i on the designated runway.

ASP aims to simultaneously assign aircraft from set J to m runways and sequence the order of the operations on each designated runway. We define a binary variable $\{z_{ir} | i \in J, r \in R\}$ for assignment of aircraft i to runway r such that $z_{ir} = 1$ if aircraft i is assigned to runway r, and $z_{ir} = 0$, otherwise. If a single runway case of ASP is under investigation (i.e., $m = 1$), then we have $\{z_{i1} = 1, \forall i \in J\}$.

8.2.1 Minimum Separation Times

Aircraft operations on the runways are required to be sufficiently separated in order to avoid the wake vortex turbulence effect. This separation is dependent on the aircraft weight class and the aircraft operation type (arrival, departure). Aircraft are

Table 8.1 Aircraft separation times (in seconds) following the FAA standard

Departure → Departure case				Departure → Arrival case			
Leading \ Following	Heavy	Large	Small	Leading \ Following	Heavy	Large	Small
Heavy	90	120	120	Heavy	60	60	60
Large	60	60	60	Large	60	60	60
Small	60	60	60	Small	60	60	60

Arrival → Departure case				Arrival → Arrival case			
Leading \ Following	Heavy	Large	Small	Leading \ Following	Heavy	Large	Small
Heavy	75	75	75	Heavy	96	157	196
Large	75	75	75	Large	60	69	131
Small	75	75	75	Small	60	69	82

categorized in different weight classes according to their maximum take off weight (MTOW). This is the maximum weight at which an aircraft type is allowed to take off. Different minimum separation standards are enforced in different regions of the world. Most commonly referenced separation times are regulated by the US Federal Aviation Administration (FAA). According to the FAA aircraft are categorized to three weight classes: Heavy, Large, and Small. Table 8.1 summarizes the minimum separation times based on the FAA regulations.

The separation times presented in Table 8.1 carry two important attributes. Firstly, these separations are asymmetric. Secondly, they do not follow the triangular inequality in certain cases, and as a result, consecutive separation times will not guarantee the proper separation of non-consecutive aircraft in the sequence. We define p_{ij} as the minimum separation time between a leading aircraft i and a following aircraft j, which depends on their operation types and weight classes and is numerically specified by a user specific standard.

8.2.2 Objective Function

In the context of scheduling problems a variety of objectives such as minimizing the makespan, maximizing throughput, minimizing total operating time, and minimization of total tardiness are commonly used. In solving ASP, other problem specific objectives are proposed and examined. Sölveling et al. (2011) studied the reduction of environmental impact. Soomer and Franx (2008) employed an equity function using airlines' preferences, and Boysen and Fliedner (2011) studied scheduling aircraft landings to balance the workload of ground staff at airports. Total cost of excess fuel burn resulted from delays was investigated in Farhadi et al. (2014).

The type of the objective function in ASP can be attractive to different parties that are involved in the process. These parties are mainly airports, airlines, and governments (Bennell et al. 2011). Therefore, the problem of managing runway

operations may require the consideration of multiple, possibly conflicting, objectives which can help reveal attractive trade-offs for the decision-maker. One of the most commonly discussed objectives in the literature is the minimization of total delays. This objective inherently has a cost reduction attribute, as minimizing the total delays will impact the direct and indirect costs of the operations for airlines and airports. It will also impact the total excess fuel burn that resulted from the delays which affects the total environmental emissions, a subject that is as of importance to the governments.

8.3 Mixed-Integer Formulations

Many studies in the literature focused on the case of scheduling landing aircraft, such as Beasley et al. (2000), Bencheikh et al. (2009), Ciesielski and Scerri (1998), and Liu (2011). On the other hand, Atkin et al. (2007) investigated sequencing departures on the runways. The mixed operation (arrival/departure) case has been explored in a number of studies such as Caprí and Ignaccolo (2004), Farhadi et al. (2014), Ghoniem et al. (2015), and Hancerliogullari et al. (2013). A case of a single runway is considered as in Atkin et al. (2007), Beasley et al. (2001), Caprí and Ignaccolo (2004), and Hu and Chen (2005). Multiple runway cases are examined in other studies, namely Hu and Di Paolo (2009), Liu (2011), and Pinol and Beasley (2006). As far as the time horizon, many focused on static set of operations, and Caprí and Ignaccolo (2004), Ciesielski and Scerri (1998), Hu and Chen (2005), and Zhan et al. (2010) demonstrated dynamic cases. To introduce the MIP formulations in this study we assume the static, multiple independent runways, mixed operation case of ASP. The variation of runway dependency is also discussed later in this section.

To formally introduce ASP as a MIP model, we consider a total weighted delay to state the objective function. The weight w_i is a user-defined parameter that can control the priority of the operations. Delays are driven by the term $t_i - r_i$. Without loss of generality, the term r_i can be eliminated and the objective function will be simplified to an equivalent objective function that minimizes the total weighted start times as depicted in Constraint (8.1a).

$$\text{Minimize} \quad \sum_{i \in J} w_i t_i \tag{8.1a}$$

We now need to assign aircraft to the runways in the set R if $(m > 1)$. Constraint (8.2a) ensures that each aircraft i from the arrival/departure set J is assigned to a runway.

$$\sum_{r \in R} z_{ir} = 1, \quad \forall i \in J \tag{8.2a}$$

Aircraft i is to be scheduled such that the time window limits are respected. Time window is formed by the earliest start time e_i, and latest start time l_i of each aircraft i. Constraint (8.3a) ensures that the time window restriction is not violated.

$$e_i \leq t_i \leq l_i, \quad \forall i \in J \tag{8.3a}$$

The constraints that are established so far are used globally in all cases of ASP. The scheduling constraints that sequence the aircraft that are assigned to the same runway will alter among different formulations of ASP. In Sects. 8.3.1 and 8.3.2 we complete modeling of ASP in three variations.

8.3.1 TSP-Based Model

Runway scheduling problem has inherent similarities to the asymmetric Traveling Salesman Problem with Time Windows (TSP-TW). In formulating the Traveling Salesman Problem (TSP) we avail of a binary variable x to construct the tours. In TSP-based model of ASP, x can be adopted to determine the consecutive precedence order of the aircraft in the sequence on each runway. Constraints (8.4a)–(8.4d) will establish this order in the problem ($J_0 \leftarrow J \cup \{0\}$: 0is a dummy node). Constraints (8.4a) and (8.4b) ensure that each aircraft has only one predecessor and one successor. Constraints (8.4c) and (8.4d) enforce assignment of only one aircraft to the beginning of the sequence, and one aircraft to the end of the sequence at all runways, respectively.

$$\sum_{j_2 \in J_0 - \{j_1\}} x_{j_1 j_2} = 1, \quad \forall j_1 \in J \tag{8.4a}$$

$$\sum_{j_1 \in J_0 - \{j_2\}} x_{j_1 j_2} = 1, \quad \forall j_2 \in J \tag{8.4b}$$

$$\sum_{j \in J} x_{0j} = m \tag{8.4c}$$

$$\sum_{j \in J} x_{j0} = m \tag{8.4d}$$

Due to the non-triangular separation times in a general case, we need an introduction of a binary variable y that specifies the overall precedence order of the sequence, holding into account the non-consecutive precedence order of the aircraft. Constraint (8.5a) identifies the aircraft that are assigned to the same runway and determines the global position of the aircraft relative to other aircraft on the same

runway. This binary variable is necessary to ensure sufficient minimum separation time between the aircraft that are not consecutively positioned in the sequence order.

$$y_{ij} + y_{ji} \geq z_{ri} + z_{rj} - 1, \quad \forall r \in R, (i,j) \in J, i < j \tag{8.5a}$$

Finally, we need to enforce the consecutive and non-consecutive separation times between the aircraft pairs on the same runway. When the order of the aircraft is identified, by enforcing the minimum separation times p_{ij} we can also compute the start time of the operations. Constraints (8.6a) and (8.6b) together apply the proper separations between every pair of aircraft and extract the corresponding start times.

$$t_j \geq t_i + p_{ij} - (1 - x_{ij})(d_i - r_j + p_{ij}), \quad \forall (i,j) \in J, i \neq j \tag{8.6a}$$

$$t_j \geq t_i + p_{ij} - (1 - y_{ij})(d_i - r_j + p_{ij}), \quad \forall (i,j) \in J, i \neq j \tag{8.6b}$$

Equations (8.1a)–(8.6b) together represent the TSP-based model of ASP. A single runway case of TSP-based ASP model is presented in Ghoniem et al. (2014). The proposed model is enhanced via efficient pre-processing, probing procedures, and valid inequalities along with development of partial convex hull representation to achieve tighter reformulations.

8.3.2 Disjunctive Models

In a disjunctive model we take advantage of the structure of the binary variable y. In the TSP formulation, the binary variable y is introduced to determine the overall position of the aircraft on the runway and enforce the non-consecutive separation times. In the disjunctive formulation of ASP, we avail of the knowledge on the aircraft position to simultaneously ensure the consecutive as well as the non-consecutive separation times.

8.3.2.1 The Case of Independent Runways

When runways are located parallel to one another and their centerlines are separated by a minimum distance of 4300 ft, they are considered independent. In this case runways can operate independently. By eliminating the TSP tour construction constraints (8.4a)–(8.4d) along with Constraint (8.6a) the TSP-based model can be transformed to a disjunctive MIP formulation. This results in a compact formulation of ASP by putting together Eqs. (8.1a)–(8.3a), (8.5a), and (8.6b). The static case of mixed operation, multiple independent runway ASP is examined in Ghoniem and Farhadi (2015) and the effects of adding valid inequalities, symmetry defeating functions, and pre-processing routines are explored.

8.3.2.2 The Case of Parallel Dependent Runways

The case of dependent runways is introduced in Beasley et al. (2000). Runways are dependent when the centerlines are moderately distanced, between 2500 and 4300 ft (runways located closer than 2500 ft operate as a single runway). In the case of dependent runways, landings and take offs on different runways need to be properly separated. We denote the separation times of aircraft on different runways by p'_{ij}.

An auxiliary binary variable χ_{ij} is introduced to ASP where $\chi_{ij} = 1$ if aircraft i and j are assigned to the same runway and $\chi_{ij} = 0$, otherwise. In order to adjust ASP to the dependent runway case the following modifications are considered. Constraint (8.5a) is adjusted to Constraint (8.7a). Constraint (8.6b) is adjusted to Constraint (8.7b). Constraints (8.7c) and (8.7d) are added to ASP.

$$y_{ij} + y_{ji} \geq 1, \quad \forall (i,j) \in J, i < j \tag{8.7a}$$

$$t_j \geq t_i + p_{ij}\chi_{ij} + p'_{ij}(1 - \chi_{ij}) - (1 - y_{ij})(d_i - r_j + \max(p_{ij}, p'_{ij})), \quad \forall (i,j) \in J, i \neq j \tag{8.7b}$$

$$\chi_{ij} \geq z_{ri} + z_{rj} - 1, \quad \forall r \in R, (i,j) \in J, i < j \tag{8.7c}$$

$$\chi_{ij} = \chi_{ji}, \quad \forall (i,j) \in J, i < j \tag{8.7d}$$

Multiple dependent runway case of ASP is presented by Eqs. (8.1a)–(8.3a) and (8.7a)–(8.7d). The binary variable y now plays the role of identifying the order of aircraft whether or not assigned to the same runway. If aircraft are on the same runway separation p will be activated, and separation p' will become active, otherwise.

8.4 Heuristic Methods

Among solution techniques advised for solving ASP, heuristic algorithms are broadly utilized either as the main approach to derive fast solutions or as an ad-hoc feature to append to an algorithm for the purpose of extracting solutions, initialization, or solution improvement. In this section we first review the heuristic approaches that utilize the relaxation of MIP models to determine the solution of ASP (Sect. 8.4.1). Other studies that propose heuristic approaches to find sub-optimal solutions based on optimization techniques are introduced in Sect. 8.4.2. Finally, in Sect. 8.4.3, a column generation heuristic is presented.

8.4.1 Generic MIP Heuristics

Multiple studies avail of the relaxation of an MIP model to explore low-cost feasible solutions to NP-hard problems which are highly costly to determine their global optimal solution. A common strategy is finding feasible upper-bounds of the MIP by problem specific heuristic techniques. A number of these strategies such as relax-and-fix or dive-and-fix techniques are based on relaxation of binary or integer variables and fixing some of the solutions to derive an upper-bound for the MIP. Further local search or exploration strategies are applied for possible improvements.

A heuristic approach to derive an upper-bound to the MIP formulation for the multiple dependent runway arrival ASP case is introduced by Beasley et al. (2000). After fixing the sequencing binary variables driven by a sorting technique, this upper-bound is used in the LP model to derive the optimal set of landing times and the total resulting cost. In this heuristic approach aircraft are sorted in the non-descending order of target times and are added sequentially to the runway that provides the best possible landing time given the preceding aircraft on the runways. Target times are a user-defined parameter that is within the time window of the aircraft and is defined as the favored time to schedule the start time of the aircraft. Any deviation from target is considered to be an earliness or tardiness.

Similarly, Soomer and Franx (2008) adopted the disjunctive MIP model for the single runway arrival ASP case. Their solution approach is grounded in the fact that if the landing sequence of the flights is given, the MIP becomes an LP formulation. A First Come First Serve (FCFS) sequence is used to initiate multiple problem specific local search heuristics. Two common local search methods, random swap and random insert, are employed to modify the initial solution in case the FCFS solution encounters infeasibility. In brief, a random swap technique is to randomly choose two members of a sequence and swap their position. A random insert technique is to remove a member within a sequence from its original position and insert it to a randomly selected position.

When the FCFS sequence is generated, an initial local search is performed to ensure feasibility of the solution. At first, new sequences are repeatedly generated, by swapping two adjacent flights for which the earlier aircraft has a larger latest landing time. Then new sequences are generated by swapping two adjacent flights that their total sequence requires less separation. Further to the initialization step, two local search schemes are introduced to improve the quality of the solution as the following.

1. Swap neighborhood: Two aircraft $(i,j) : i < j$ in the sequence are selected to swap positions if $l_i > e_j$. If there is an aircraft $k : i < k < j$ where $e_k < l_i$ but $e_k \leq l_j$, swapping i and j will cause infeasibility. Therefore, swap $(i, j-1)$ and (k, j).
2. Shift neighborhood: Remove aircraft i from it's position and insert in aircraft j's position, where $i < j$ and $l_i > e_j$. If there is an aircraft $k : i < k < j$ where $e_k < l_i$ but $e_k \leq l_j$, insert i into position of aircraft $j-1$ and insert k into position of j.

After each iteration of local search, binary sequence variables y_{ij} are fixed, therefore ASP becomes an LP. It has been shown that the schedules obtained by the proposed methodology, yield tremendous savings compared to the FCFS with an average of 33 % reduction in costs (Soomer and Franx 2008).

8.4.2 Optimization-Based Heuristics

Optimization-based heuristic methods are based on an optimization technique of MIP with the addition of a single or multiple restrictions to limit the solution space. These restrictions are introduced to the MIP as additional constraints which prune the feasible space and restricts the solution space to a set of sub-optimal solutions to the original MIP. In the literature it has been shown that although this technique does not guarantee the optimal solutions, it enables the delivery of near-optimal solutions with minimal computational effort.

For the case of single runway mixed operations, Ghoniem and Farhadi (2015) proposed two heuristics that rely on the optimization MIP models of ASP and compared the solutions with the traditional FCFS policy. The proposed heuristics are as follows.

8.4.2.1 Optimized FCFS

This heuristic produces two FCFS sequences of arrivals and departures so that no aircraft in the same queue (arrival or departure) is overtaking another. However, the interweaving of the queues are sequenced optimally, given the time windows and minimum separation times. The heuristic can be applied by appending Constraint (8.8) to the model ASP.

$$t_i + p_{ij} \leq t_j, \quad \forall (i,j) \in J, i \neq j | e_i < e_j, \text{and } (i,j) \in \text{same queue} \tag{8.8}$$

8.4.2.2 Threshold-Based Sub-optimized Heuristic

The Threshold-based heuristic is a relaxed version of optimized FCFS that aims to limit the solution space by forcing a FCFS sequence to the aircraft on the same queue if their earliest start time are separated by more than a user-defined parameter δ. This heuristic is applicable by appending Constraint (8.9) to the single runway case of ASP model.

$$t_i + p_{ij} \leq t_j, \quad \forall i,j \in J, i \neq j | e_i + \delta < e_j, \text{and } (i,j) \in \text{same queue} \tag{8.9}$$

Optimized FCFS and the Threshold-based heuristic algorithms showed 4.3 % and 5 % improvement in the makespan relative to the FCFS order with landing priority when tested over simulated instances (Ghoniem et al. 2014). Along the same line, Farhadi et al. (2014) designed two optimization-based heuristics for the mixed operation, multiple independent runway ASP. The heuristics are described as follows.

8.4.2.3 Optimal Runway Assignment with Joint-FCFS Sequence

This heuristic ranks aircraft based on their earliest start times e_i, and assigns them to the first available runway. Under this strategy, no aircraft is allowed to overtake an earlier aircraft in the sequence. This heuristic can be implemented by appending Constraint (8.10) to the multiple runway case of ASP.

$$t_i \leq t_j, \quad \forall (i,j) \in J, i \neq j | e_i < e_j \tag{8.10}$$

8.4.2.4 Optimal Runway Assignment with Disjoint-FCFS Sequence

Under this heuristic, the assignment of aircraft to runways is optimized with the restriction that no aircraft can overtake aircraft with the same operation type. However, position shifts are allowed among aircraft from opposite operation type under their time window limits. As a result, it generates a FCFS sequence within each operation type queue but not across the queues. The proposed heuristic can be implemented using the multiple runway ASP and by enforcing Constraint (8.11).

$$y_{ji} = 0, \quad \forall (i,j) \in J, i \neq j | e_i < e_j \text{ and } (i,j) \in \text{same queue} \tag{8.11}$$

The optimization-based heuristics proposed by Farhadi et al. (2014) showed very near-optimal and optimal solutions when minimizing total fuel cost with an average position shift of 2 per shifted aircraft in the original sequence.

8.4.3 Column Generation

ASP can be reformulated as a set partitioning model where each column represents the set of aircraft that are assigned to the same runway. The following restricted master problem, denoted by **RMP**, provides an alternative formulation for the ASP where Q^h is the column representing the subset of aircraft sequenced on the same runway at a cost c_h. Binary variable u_h is the decision variable associated with each column Q^h. H denotes the total number of columns in hand.

$$\textbf{RMP: Minimize} \quad \sum_{h=1}^{H} c_h u_h \tag{8.12a}$$

$$\text{subject to} \sum_{h=1}^{H} Q_i^h u_h = 1, \quad \forall i \in J \tag{8.12b}$$

$$\sum_{h=1}^{H} u_h = m \tag{8.12c}$$

$$u \text{ binary.} \tag{8.12d}$$

Solving the relaxation of RMP returns the shadow prices of the current restricted master problem with the limited on hand columns. We need a procedure to construct new columns given the updated dual values at each iteration. To this end, we need to formulate a subproblem that finds optimal or near-optimal columns with minimum reduced costs. Two approaches are advised: (a) Construction of a MIP formulation for solving the subproblem via optimization techniques and (b) utilizing dynamic programming to find the best minimum reduced cost column.

8.4.3.1 Optimization-Based Subproblem

We can formulate the subproblem as an MIP problem. Consider the following notation:

- $\zeta_i \in \{0, 1\}$: $\zeta_i = 1$ if and only if aircraft i is selected in the column constructed by the subproblem, $\forall i \in J$.
- π: vector of dual variables of the Constraint (8.12b), where $\pi = \bar{\pi}$ represents specific dual values obtained at a given iteration of column generation.
- π_0: dual variable of the Constraint (8.12c), where $\pi_0 = \bar{\pi}_0$ represents a specific dual value obtained at a given iteration of column generation.

The subproblem, denoted by $\mathbf{SP}(\bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\pi}}_{\mathbf{0}})$, is defined as follows:

$$\mathbf{SP}(\bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\pi}}_{\mathbf{0}})\textbf{: Minimize} \quad \sum_{i \in J} (w_i t_i - \bar{\pi}_i \zeta_i) - \bar{\pi}_0 \tag{8.13a}$$

$$e_i \zeta_i \le t_i \le l_i \zeta_i, \quad \forall i \in J \tag{8.13b}$$

$$t_j \ge t_i + p_{ij} y_{ij} - (1 - y_{ij}) \max_{i \in J} \{l_i\}, \quad \forall (i, j) \in J, i \ne j \tag{8.13c}$$

$$y_{ij} + y_{ji} \ge \zeta_i + \zeta_j - 1, \quad \forall (i, j) \in J, i < j \tag{8.13d}$$

$$y_{ij} + y_{ji} \le \zeta_i, \quad \forall (i, j) \in J, i \ne j \tag{8.13e}$$

$$y, \zeta \text{ binary.} \tag{8.13f}$$

The objective function (8.13a) minimizes the reduced cost of the column that is being constructed by the pricing subproblem. Constraint (8.13b) enforces a time window on aircraft that are included in the column. Constraint (8.13c) enforces the separation time between aircraft. Constraints (8.13d)–(8.13e) ensure that only aircraft that belong to the column are sequenced. In this context, Ghoniem et al. (2015) introduce accelerating schemes to the column generation iterative process; namely, complementary column generation, and interior point stabilization. They show that under manageable instance sizes the optimization-based subproblem can produce very near-optimal solutions with improvement in the computation time compared to the MIP formulation.

8.4.3.2 Dynamic Programming

Ghoniem et al. (2015) demonstrated a dynamic programming scheme to solve the pricing subproblem of the column generation approach for ASP. Noting the similarity between the subproblem of ASP and the Vehicle Routing Problem with time-windows (VRPTW), problem SP is equivalent to solving an elementary shortest path problem. Each aircraft is referred to as a node. Every pair of nodes is connected via a directed arc whose cost equals to the minimum separation time between the corresponding pair of aircraft.

The elementary shortest path problem seeks to find a shortest path to visit a number of nodes by extending paths from one node to the other reachable nodes. These paths are referred to as labels as they carry information about the cost of the path, nodes visited, and resources consumed. Dominance rules are employed to eliminate the inferior labels at each iteration. Each node appears on a path at most once. The algorithm terminates when no new label is available.

Path Extension and Dominance Rules

In the algorithm a dummy origin o is considered where $c_o = -\bar{\pi}_0$ and $t_o = 0$. A path is extended from the origin to all other reachable aircraft. If we extend a path ρ from origin to aircraft i then we have $c_i(\rho) = -\bar{\pi}_0 + (w_i t_i - \bar{\pi}_i)$. The operation time t_i will be computed according to the e_i and the minimum separation time with the preceding nodes. Reachable aircraft from i are updated so that in case of extension of the path to the available aircraft the time window restrictions are not violated.

When two distinct paths are extended to a particular aircraft, the dominance rule will be applied to eliminate the inferior path. If no path is proven to be inferior, all paths will be kept at each iteration. This step is necessary to reduce the size of the problem and increase the tractability of the paths. At node i a path ρ_1 is inferior to ρ_2 if and only if all of the following criteria apply: (a) $c_i(\rho_1) \geq c_i(\rho_2)$, (b) $t_i(\rho_1) \geq t_i(\rho_2)$, and (c) $E_i(\rho_1) \subseteq E_i(\rho_2)$, where $E_i(\rho_1)$ is the set of reachable aircraft of path ρ_1 from aircraft i.

It is worth noting that generic dynamic programming schemes for shortest path problems base the procedure on the assumption that triangular inequality holds for arc costs. However, in ASP if FAA separation times are utilized the dominance rules will no longer apply. To this end, Ghoniem et al. (2015) form a modification to the dominance rules by identifying exceptions in the calculation of the path costs to cope with the non-triangular values in separation times. In the study it is shown that dynamic programming scheme has great computational advantage over the MIP model.

8.5 Meta-heuristics

In this section we present the meta-heuristic algorithms studied in the literature to solve various cases of ASP. Section 8.5.1 describes a generic scheme of genetic algorithms and discusses a number of studies that utilize this approach.

8.5.1 Genetic Algorithms

One common approach in the literature to solve ASP is genetic algorithms (GA). Many variations of GA have been designed and studied with different approaches towards modeling the problem under the GA setting, with variety of GA operators which are used to update the solution space. In GA a pool of solutions are produced. Each solution is referred to as a chromosome. The chromosome consists of a series of genes that hold information about the solution. An objective function of choice is designed to evaluate the fitness of each chromosome. Probability functions are used to randomly select fitter chromosomes. The selected chromosomes (parents) are combined by a rule defined by a crossover operator. Parents mate based on the random functions in the crossover operator to give birth to a single or multiple offspring (children). Offspring will have a fitness value of their own. The quality of the offspring are improved by a mutation operator. Offspring are compared to the current generation and the inferior chromosomes are eliminated. This procedure is continued until no improvement can be achieved in the fitness or a termination criterion is met. A general scheme of a GA operation is depicted in Algorithm 1 as follows.

The quality and the convergence of the GA depends highly on the design of the chromosome (encoding phase) and the design of the random operators. A variety of approaches are used to encode ASP for GA. The approaches vary for single runway case and multiple runway case. In this section we review multiple studies that take different approaches towards the GA implementation of ASP.

Beasley et al. (2001) take a different approach towards ASP. Encoding is based on a proportion value θ_j for aircraft j which defines the start time of the aircraft based on the percentage distance from the ready time [Eq. (8.14a)]. The fitness

Algorithm 1 Genetic algorithm scheme

```
Generate the initial population
Compute the fitness
while Termination criterion is not met do
    Parent selection operator: Randomly select parents based on relative fitness
    Crossover operator: Mate the parents and create offspring
    Mutation operator: Improve the quality of the offspring
    Population replacement
end while
```

function is a nonlinear squared earliness or tardiness. A measure of unfitness is also presented relative to the violation from the separation times and infeasibility. Parent selection is a binary tournament selection. Crossover operator is a uniform crossover function where the genes in the child are taken from one or other parent chosen at random. Population replacement makes use of the unfitness and fitness values where it eliminates the inferior member based on fitness and unfitness. The GA was tested on an instance size of 20 aircraft driven from real life airport observation. Average delays were decreased by 2.16 % relative to the actual operating times.

$$t_j = r_j + \theta_j(d_j - r_j) \tag{8.14a}$$

A different strategy has been devised by Liu (2011) to encode the chromosomes on the multiple runway case. In this study, a chromosome is a string with the length of arrival queue. The genes imply the runway index that is assigned to the corresponding aircraft. Runway assignment is random. The fitness function is the squared deviation of landing time from the earliest possible landing time. Mutation is a random swap of the genes, which are the designated runways.

8.5.1.1 GA for the Dynamic Case

It is worth adding that GA also has been employed to solve the ASP under dynamic environment where the problem set changes dynamically over time as the new aircraft enters the queue of arrivals or departures and the aircraft that already operated are removed from the queue. Hu and Chen (2005) introduced the Receding Horizon Control (RHC) to the single runway arrival ASP. Receding horizon consists of a number of time intervals. A time interval is a randomly selected constant value that divides the time space into uniformly distant spaces. The algorithm performs on a receding horizon. The decisions are fixed in the first time horizon as it exits the receding horizon and the new time interval enters, sequentially.

In this study, they encoded each chromosome by defining a possible landing sequence. The fitness function is the airborne delay of the possible landing sequence and a terminal penalty, which assesses the influence of the current landing sequence on those aircraft outside the receding horizon. The mutation operator is a random swap of genes adjusted in a way that the closer pairs of aircraft are more likely

to swap positions. Hu and Di Paolo (2009) extended the GA with RHC to the multiple runway case. The encoding system is based on a n^2 matrix system where the diagonal values represent the runway assignment and other entries define the aircraft sequence.

In contrast, Ciesielski and Scerri (1998) used an 8 bits chromosome that represents a landing time (7 bits) and a runway. The study defined time intervals and in the encoding procedure deletes planes that have landed and adds planes that have recently arrived. In another study by Caprí and Ignaccolo (2004), a dynamic environment for the single runway arrival case is investigated. In this paper, each chromosome encodes the aircraft index in the order of the arrival queue. Crossover operates on genes that are identified by a randomly generated binary string consisting of 0,1 values. They utilized that in two ways: either move the genes of a parent matching the position of 1 values on the string, or fix the parent genes matching the 1 values and move the rest.

8.5.2 Ant Colony Optimization

ASP can be formulated as an ant colony optimization problem (AC). AC is a probabilistic meta-heuristic approach which was originally introduced to solve shortest path problem in a graph. In AC a solution is constructed based on the random moves of an ant that leaves the origin and selects to traverse an edge to get to the next node based on a random state transition rule. The state transition rule is composed of two values, the attractiveness and the pheromone level. State transition rule is defined as Eq. (8.15a). Under this rule if a random value q is less than a predefined random value q_0 then the edge with the highest combination of priority and pheromone level will be chosen. An edge will be randomly selected based on the probability values P_{ij} otherwise. The probability rule to choose the edges is calculated by Eq. (8.15b). α and β are control parameters.

$$(i,j) = \begin{cases} \arg\max_{k \in \text{allowed}_j} \left\{ \tau_{ij}^{\alpha} \eta_{ij}^{\beta} \right\} & q \leq q_0 (\text{exploitation}) \\ \text{Select randomly by } Pr_{ij} & \text{otherwise (exploration).} \end{cases} \tag{8.15a}$$

$$P_{ij} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{k \in \text{allowed}_j} \tau_{ik}^{\alpha} \eta_{ik}^{\beta}} \tag{8.15b}$$

The attractiveness (priority) of the edge η_{ij} is calculated by a problem specific heuristic. The trail pheromone level τ_{ij} starts at an initial value and is updated at each iteration by a pheromone evaporation coefficient and the quantity of pheromone left on the trail by the colony. The pheromone updating rule is defined as Eq. (8.16a). φ is the pheromone evaporation coefficient. Equation (8.16b) calculates the amount of pheromone deposited on an edge after all ants completed the solution. Θ is a

Algorithm 2 Ant colony optimization scheme

```
Initialize the matrix of τ_ij and η_ij
Initialize state transition probabilities P_ij
for Each ant in the colony do
    Select starting point
    while List of candidate edges is not empty do
        Select edges according to state transition rule
    end while
    Break if the termination criterion is met
    Update the pheromone trail
    Update the state transition rule
end for
```

constant and C can be selected heuristically which is mostly the cost of the solution in the given iteration. The cost of the generated solutions can be evaluated with an objective function of choice.

$$\tau_{ij} \leftarrow (1-\varphi)\tau_{ij} + \Delta\tau_{ij} \tag{8.16a}$$

$$\Delta\tau_{ij} = \begin{cases} \Theta/C & \text{if edge } (i,j) \text{ is visited} \\ 0 & \text{otherwise.} \end{cases} \tag{8.16b}$$

The general scheme of AC algorithm is depicted in Algorithm 2.

Bencheikh et al. (2011) adapted AC for the multiple runway landing ASP case. They present a bi-level graph. In the first level, the available runways are selected and in the second level the aircraft are determined. The probability rule for runway selection mechanism is according to the availability time to receive new aircraft. Aircraft selection follows the state transition rule where the priority information is driven from the earliest landing time, target time, and the corresponding cost penalty.

Zhan et al. (2010) extended the static single runway arrival ASP to a dynamic case using RHC. Algorithm is performed on the entire horizon, but only decisions made for the first time interval are fixed. In another study, Bencheikh et al. (2009) used AC in a hybrid algorithm. AC is utilized to produce the initial population, and feed a GA that updates the solution pool derived by the AC algorithm. In this study, priority rules depend on two parameters: separation times and the difference of the aircraft target times. The improved AC algorithm is tested on the small benchmark instances of OR-Library and produces optimal and near-optimal solutions.

8.5.3 Simulated Annealing

Simulated annealing (SA) is a direct probabilistic meta-heuristic with no memory. At each iteration, SA considers some neighbors of a current state, and decides between moving the system to the new state or staying in the same state. Since

Algorithm 3 Simulated annealing scheme

```
Generate a random solution s
Calculate the current state cost f
while Termination criterion is not met do
    Generate a random neighborhood s'
    Calculate the new state cost f'
    if f' better than f then
        Accept f'
    else
        Let Δf = f' − f
        if e^{−Δf/T} > Random(0, 1) then
            s ← s'
        end if
    end if
end while
```

SA directly evolves from an initial solution, the quality of the final solution highly depends on the initial solution construction scheme and the design of the local search schemes. When better solutions are generated, SA accepts the new state directly. However, SA also allows inferior solutions to replace the current state with a probability rule. This feature enables SA to escape local optimal solutions. The probability of accepting a move, which causes a change Δf in the objective function, is called the acceptance function and is depicted in Eq. (8.17a).

$$e^{-\Delta f/T} > \text{Random}(0, 1) \tag{8.17a}$$

T is a control parameter corresponding to the temperature in the analogy with physical annealing. When T is high, most moves will be accepted, but as T approaches zero most moves will be rejected unless they produce better objective function values. Usually, the SA begins with high values of T in order to avoid local optimum solutions. The temperature gradually drops while the algorithm reaches the final iterations (cooling process). The SA scheme can be summarized as in Algorithm 3.

Salehipour et al. (2013) introduced a SA algorithm to solve the multiple runway arrival ASP. The initial solution is constructed by sorting aircraft by ready times and assigning consecutive aircraft to different runways if their distance is less than their separation time. Aircraft are assigned to the same runway of otherwise. They used a variable neighborhood descent and a variable neighborhood search for intensifying and diversifying the solution. Three improvement rules are implied. A random swap within a runway, random swap among runways, and a random insert which removes aircraft from one runway and inserts to another. SA algorithm offered in this study is tested on the OR-Library instances and demonstrates fast convergence to optimal solutions with short computation time on the instances with 50 aircraft and less. For most of the larger instance sizes the SA converges within a manageable time and the quality of the solutions are comparable to that of a Cplex commercial solver with 1000 s of CPU time.

Algorithm 4 Meta-RaPS scheme

```
Calculate the priority value for all elements i in the set
for I iterations do
    while Set of candidates is not empty do
        Select a candidate i to add to the solution
        if Random(0, 1) < pp% then
            Add i to the solution using a greedy heuristic
        else
            Randomly select j from the candidate list where priority_i ≥ (priority_best)rp%
            Add j to the solution using a greedy heuristic
        end if
    end while
    Calculate the fitness of the solution f
    if f < f_best + (f_worst − f_best)ip% then
        Implement a local search heuristic to improve f
    end if
    Record the solution
end for
Report the best solution
```

In a similar work, Fahle et al. (2003) presented an SA algorithm for single runway arrival ASP where a solution is represented only by the order in which the planes should land. All possible permutations are allowed. A penalty function for infeasible (illegal) solutions is added to the algorithm to avoid infeasible states.

8.5.4 Meta-RaPS

Meta-heuristic for Randomized Priority Search (Meta-RaPS) has two main stages: a solution constructor, that generates solutions using a greedy heuristic, and a local search heuristic, that improves the solutions. The Meta-RaPS algorithm consists of four parameters: number of iterations I, the priority percentage $pp\%$, the restriction percentage $rp\%$, and the improvement percentage $ip\%$. For I iterations, Meta-RaPS constructs feasible solutions, improves them through a local search heuristic, and selects the best solution upon termination. During each greedy solution construction phase, the parameter $pp\%$ is used to determine whether to add the next best element according to its priority value to the solution. The priority value is calculated by a user-defined method for all elements of the set. If the priority rule fails, the next element is randomly chosen from the feasible elements whose priority values are within $rp\%$ of the best priority value. After completing the solution construction, local search heuristic is applied for improvement if the fitness of the solution is within the $ip\%$ of the best fitness value found so far. A summarized scheme of Meta-RaPS is shown in Algorithm 4.

Meta-RaPS is applied to the case of multiple runway mixed operations ASP in a study by Hancerliogullari et al. (2013). In this paper three heuristics are proposed

to calculate the priority values when selecting candidate elements of the set. The first heuristic resembles the FCFS rule. In the second heuristic a priority value is measured by a weighted factor of earliest time, target time, latest time, and the separation time. The third heuristic combines these parameters in an exponential manner. A simulated annealing algorithm is also designed to be examined along with the Meta-RaPS. It is observed that over randomly generated instances of up to 25 aircraft and 5 runways optimal and near-optimal solutions are achieved with minimal computational effort. The quality of the proposed SA is relatively better than the Meta-RaPS when the same greedy algorithms are considered for initial solutions.

8.5.5 Tabu Search

Tabu search (TS) is an extension of the local search methods that keeps short memory. It has been developed to hedge against one of the short comings of local search algorithms, getting stuck in a local optimal solution. Similar to SA, TS has a feature to avoid similar circumstances. In a TS algorithm, an initial solution is generated and a number of the neighbors of the solution are explored by random local search heuristics such as swap, or insert. The neighbors are examined for their quality and feasibility. The best solution is selected to become the next initial solution. Moreover, a list of the movements are recorded, including the movements that resulted infeasibility. This list is defined as the Tabu list and will prohibit the local search from making the exact moves again. Older memories will be discarded after an appointed time. This feature enables TS from avoiding cycles and local optima. The TS scheme is depicted in Algorithm 5.

Atkin et al. (2007) designed and implemented a TS algorithm for a single runway departure ASP. In this study, they introduced the concept of holding patterns to the departure scheduling. Holding points are entrance queues to the departure stream. Aircraft go through holding points to get to the runways. An allocation heuristic is

Algorithm 5 Tabu search scheme

```
Initiate the Tabu list
Generate a random solution s
Calculate the current state cost f
while Termination criterion is not met do
    Generate a set of random neighborhoods of s
    Update the Tabu list
    Eliminate the neighbors that include Tabu movements
    Find the best neighbor s' with minimum cost f'
    if f' < f then
        s ← s' and f ← f'
    end if
end while
```

Algorithm 6 Scatter Search and Bionomic Algorithm scheme

```
1: Generate the initial population (Reference set)
2: Improve all members through local search
3: while Termination criterion is not met do
4:     Elite set selection: Select a subset of the best members from the Reference set
5:     Solution combination: Generate new solutions from the members of the Elite set
6:     Improve all new members through local search
7:     Update the Reference set
8: end while
```

designed so that it allocates slower paths to aircraft that are overtaken and faster paths to aircraft that overtake. By this heuristic aircraft in the departure stream can change positions. For initialization, aircraft are in the order at which they arrived at the holding points. Several random swapping strategies are developed to explore the solution space. In this study the best feasible candidate neighbor that does not involve a tabu move is set as the new initial solution even if it is inferior to the last initial solution. The algorithm is tested on real time data sets consisting approximately 300 aircraft. Significant improvements are achieved in the total cost and total delay compared to the actual order and actual times.

8.5.6 Scatter Search and Bionomic Algorithm

Most of the population heuristic algorithms such as GA utilize random operating procedures to execute different phases of their iterative search, namely the initialization, exploration and exploitation, parent selection, and generating new solutions. Scatter Search (SS) and Bionomic Algorithm (BA) use discrete operators for selecting good candidates and generating new solutions based on the current state of the population. A general description of SS and BA is depicted in Algorithm 6.

In the SS and the BA initialization phase generation of the first population is random with the inclusion of seed solutions. In the subset selection phase (parent selection) two or more candidates are selected among the best members of the reference set. These best candidates are not necessarily the fittest in terms of their objective value. If a member improves the diversity of the reference set it may be considered a best candidate. Solution combination (parent mating) phase is structured as a convex or non-convex combination of solutions in the selected subset. The local improvement procedure is applied to all individuals. SS and BA attempt to maintain a diverse set of high quality solutions (the elite set) at all iterations and generate weighted centers of selected sub-regions.

Pinol and Beasley (2006) examined SS and BA on the multiple runway arrival ASP. In the SS algorithm, parent selection procedure is a binary tournament scheme based on individual fitness. The number of the parents is set to be three. The presented BA utilizes a distance measure in addition to the fitness value for construction of the elite set. This distance measure for every pair of aircraft is set

to 1 if aircraft are on different runways and is computed as the absolute difference of the proportion values (see Sect. 8.5.1) of the pair otherwise. Solutions from the reference set which have greater distance are considered for the elite set. This strategy ensures the diversification of the BA algorithm. A convex combination of members of elite set is used to generate new members. The weights are fixed randomly. The SS and the BA achieved 3.4 % and 2.9 % gap, respectively, to the best known solution for the large instances of OR-Library with a nonlinear objective function.

8.6 Conclusion and Further Remarks

This paper addressed heuristic and meta-heuristic algorithms for solving ASP under variety of settings. ASP can be classified under several distinguishing parameters of the problem such as number of the runways (single/multiple), type of the runway setting (dependent/independent), type of the operations on the runway (arrival/departure/mixed-operation), and type of the decision horizon (static/dynamic). A general MIP structure for ASP can be modeled based on a TSP, or by presenting a disjunctive variable to determine the order of the aircraft on the designated runways in a general mixed-operation multiple runway setting. This problem also can be reformulated as a set partitioning problem that requires a column generation scheme to explore the solution space. ASP aims to simultaneously assign aircraft to an available runway and sequence the operations so that the time window restrictions are not violated and minimum separation times are respected.

ASP is in the class of NP-hard problems and therefore an optimization procedure to find the global optima can be computationally burdensome. Several heuristic and meta-heuristic schemes, with greater attention to arrival scheduling under static decision space, has been proposed in the literature. A number of studies proposed random local search heuristics that are initiated from heuristic or greedy sorting algorithms such as FCFS. Utilization of MIP model and generation of sub-optimal solutions by appending constraints to prioritize the operations was found to be an efficient approach to find very near-optimal solutions with minor position shifts from the original order.

As far as the meta-heuristic schemes, genetic algorithms have been widely used for both cases of static and dynamic ASP. Different approaches towards encoding the problem have been advised. However, the random operators are commonly shared, with minor adjustments. Generally, the genetic algorithms are observed to be efficient in finding good quality solutions within a manageable time. Simulated annealing, Meta-RaPS, and Tabu search meta-heuristics are the next common approaches for solving ASP. The computational time and solution quality driven from these algorithms are competitive. As it has been examined,

substantial improvements are achievable in delay and cost reduction compared to the commonly applied FCFS order. The quality of the solution in these algorithms is highly impacted by the parameter calibration.

Further investigation of branching schemes and utilization of hybrid meta-heuristic algorithms with more emphasis on departure scheduling and dynamic decision space is suggested.

References

Atkin JA, Burke EK, Greenwood JS, Reeson D (2007) Hybrid metaheuristics to aid runway scheduling at London Heathrow airport. Transp Sci 41(1):90–106

Beasley JE, Krishnamoorthy M, Sharaiha YM, Abramson D (2000) Scheduling aircraft landings—the static case. Transp Sci 34(2):180–197

Beasley JE, Sonander J, Havelock P (2001) Scheduling aircraft landings at London Heathrow using a population heuristic. J Oper Res Soc 52:483–493

Bencheikh G, Boukachour J, Alaoui AEH, Khoukhi FE (2009) Hybrid method for aircraft landing scheduling based on a job shop formulation. Int J Comput Sci Netw Secur 9(8):78–88

Bencheikh G, Boukachour J, Alaoui AEH (2011) Improved ant colony algorithm to solve the aircraft landing problem. Int J Comput Theory Eng 3(2):224–233

Bennell JA, Mesgarpour M, Potts CN (2011) Airport runway scheduling. 4OR 9(2):115–138

Boysen N, Fliedner M (2011) Scheduling aircraft landings to balance workload of ground staff. Comput Ind Eng 60(2):206–217

Caprí S, Ignaccolo M (2004) Genetic algorithms for solving the aircraft-sequencing problem: the introduction of departures into the dynamic model. J Air Transp Manag 10(5):345–351

Ciesielski V, Scerri P (1998) Real time genetic scheduling of aircraft landing times. In: Proceedings of the 1998 IEEE international conference on evolutionary computation. IEEE World Congress on Computational Intelligence. IEEE, New York, pp 360–364

Fahle T, Feldmann R, Gtz S, Grothklags S, Monien B (2003) The aircraft sequencing problem. In: Computer science in perspective. Springer, Berlin/Heidelberg, pp 152–166

Farhadi F, Ghoniem A, Al-Salem M (2014) Runway capacity management – an empirical study with application to Doha International Airport. Transp Res E Logist Transp Rev 68:53–63

Ghoniem A, Farhadi F (2015) A column generation approach for aircraft sequencing problems: a computational study. J Oper Res Soc 66:1717–1729

Ghoniem A, Sherali HD, Baik H (2014) Enhanced models for a mixed arrival-departure aircraft sequencing problem. INFORMS J Comput 26(3):514–530

Ghoniem A, Farhadi F, Reihaneh M (2015) An accelerated branch-and-price algorithm for multiple-runway aircraft sequencing problems. Eur J Oper Res 246:34–43

Hancerliogullari G, Rabadi G, Al-Salem AH, Kharbeche M (2013) Greedy algorithms and metaheuristics for a multiple runway combined arrival-departure aircraft sequencing problem. J Air Transp Manag 32:39–48

Hu XB, Chen WH (2005) Genetic algorithm based on receding horizon control for arrival sequencing and scheduling. Eng Appl Artif Intell 18(5):633–642

Hu XB, Di Paolo E (2009) An efficient genetic algorithm with uniform crossover for air traffic control. Comput Oper Res 36(1):245–259

Liu YH (2011) A genetic local search algorithm with a threshold accepting mechanism for solving the runway dependent aircraft landing problem. Optim Lett 5(2):229–245

Pinol H, Beasley JE (2006) Scatter search and bionomic algorithms for the aircraft landing problem. Eur J Oper Res 171(2):439–462

Salehipour A, Modarres M, Naeni LM (2013) An efficient hybrid meta-heuristic for aircraft landing problem. Comput Oper Res 40(1):207–213

Sölveling G, Solak S, Clarke JPB, Johnson EL (2011) Scheduling of runway operations for reduced environmental impact. Transp Res Part D Transp Environ 16(2):110–120

Soomer MJ, Franx GJ (2008) Scheduling aircraft landings using airlines preferences. Eur J Oper Res 190(1):277–291

Zhan ZH, Zhang J, Li Y, Liu O, Kwok SK, Ip WH, Kaynak O (2010) An efficient ant colony system based on receding horizon control for the aircraft arrival sequencing and scheduling problem. IEEE Trans Intell Transp Syst 11(2):399–412

Chapter 9
A Tabu Search Algorithm for the Multiple Runway Aircraft Scheduling Problem

Bulent Soykan and Ghaith Rabadi

Abstract Runways are typically identified as the primary bottleneck of the airport operations system that causes delays. Hence, operational efficiency of runways constitutes a critical factor for the overall air transportation system. Multiple Runway Aircraft Scheduling Problem involves assigning both landing and taking-off aircraft to runways, sequencing them on each runway and assigning each aircraft a landing or take-off time while considering predetermined time windows for each aircraft to land or take-off. Also, sequence-dependent separation times between each aircraft pair in the sequence need to be taken into account in order to avoid wake vortex (turbulence) effects which can pose a hazard caused by preceding aircraft. Several variations of this combinatorial optimization problem are researched extensively in the past decades and a wide variety of algorithms have been proposed for small-scale problems. However, from a practical point of view large-scale real-life problems require fast response times and remain challenging computationally. This chapter aims to present a Tabu Search (TS) algorithm for the static (offline) case of the problem, where all information of aircraft is known in advance. Also, computational results for the proposed algorithm are presented for a number of benchmark instances obtained from literature.

Keywords Aircraft scheduling • Runway scheduling • Tabu search • Metaheuristics

9.1 Introduction

As the demand for air transportation continues to increase throughout the world, the air traffic volume in major airports approaches to the capacity of the airport infrastructure. Runways are typically identified as the primary bottleneck in the airport infrastructure that causes delays and the capacity of an airport heavily

B. Soykan • G. Rabadi (✉)
Department of Engineering Management and Systems Engineering, Old Dominion University, Engineering Systems Building, Rm 2102, Norfolk, Virginia 23529, USA
e-mail: grabadi@odu.edu

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_9

depends on the runways in use. Therefore, high volume air traffic results in runway congestion, and consequently long queues for land and take-offs, fuel costs, and environmental impacts. Although one may think that investing in airport infrastructure can solve the problem, most of the time it is neither practical nor feasible. The lack of physical space for new runways at most major airports coupled with newly promulgated environmental restrictions prevent adding more runways to increase the capacity. As a result, it is significantly important to effectively and efficiently utilize runways to increase the overall capacity of the airports and to smooth the flow of air traffic. To accomplish this, fast and efficient algorithms are required for scheduling aircraft over runways as part of decision support systems used by air traffic controllers.

The Multiple Runway Aircraft Scheduling Problem (MRASP) is a decision-making problem in which aircraft must be assigned to runways, sequence the assigned aircraft on each runway and then determine each aircraft's start time (landing or take-off). The objective of the problem in this study is to minimize the total tardiness while considering certain operational restrictions, such as time windows and separation requirements. Once each aircraft enters the radar range for landing or pushbacks from the gate for take-off, air traffic controllers assign a runway and a start time to land/take-off. The start time has to be between predetermined earliest and latest land/take-off time. Also there is a target time to land/take-off within this time window, which is the time that aircraft can land if it flies at its cruise speed for landing and the most probable time for take-off considering the taxi and holding times for take-off. Separation requirements between both landing and taking-off aircraft, which make the problem a non-trivial one, need to be taken into account due to the safety reasons associated with wake turbulence. Rigorously scheduling the aircraft over runways have potential to reduce the sum of all separation times between aircraft.

Considering its similarities to production scheduling problems, the MRASP is usually mapped to identical parallel machine scheduling problem with sequence-dependent setup times problem, which is *NP*-Hard for most objective functions (Garey and Johnson 1979). The problem can be formulated as a mixed integer programming (MIP) model, a set partitioning model or as asymmetric traveling salesman problem with time windows. Due to problem complexity, exact solution methods are in general not capable of solving practical problem sizes. One of the main alternative solution methods for solving *NP*-Hard problems is metaheuristics. Given computational complexity of the MRASP, a Tabu Search (TS) metaheuristic algorithm, is proposed to find solutions of the problem.

The overall aim of this chapter is to solve the problem of combined aircraft arrival/departure aircraft scheduling problem over multiple independent runways using a TS algorithm. The effectiveness of the proposed approach is tested on a number of benchmark instances obtained from literature. To the best of our knowledge, this is the first attempt in the literature to apply TS method to MRASP.

The remainder of the chapter is organized as follows. Next section provides a formal description of the MRASP, including the mathematical formulations. Section 9.3 describes an overview of what was gleaned from the literature review.

Section 9.4 explains the underlying principles of the TS algorithm developed for the MRASP. Section 9.5 summarizes the experimental design and the results of the numerical experiments. Section 9.6 contains the summary and conclusions including plans for future work.

9.2 Description and Formulation of the Problem

MRASP is a large-scale scheduling problem that consists of a three-step process. The first step involves allocating aircraft to different runways, the second includes sequencing the aircraft allocated to each runway, and the third step is to determine the start time for each aircraft. This problem arises usually at busy airports where runway utilization needs to be optimized to prevent delay related costs. Given a number of aircraft with their time windows for land or take-off and separation times required for each pair of aircraft and a set of runways, the objective of MRASP is to minimize the total weighted delay (or tardiness), which is calculated as the non-negative difference between the start time and the target land/take-off time for each aircraft.

The start time of each aircraft depends on a predetermined interval called time window, constrained by an earliest and latest land/take-off time. The earliest land time corresponds to the time at which the aircraft could land if it uses its fastest speed. Within the time window, there is a target time which relates to the time that aircraft could land if it flies at the most economical speed.

The other major operational constraint that needs to be taken into account is the problem of wake vortex (turbulence), which is associated with aircraft performance characteristics. Aircraft passing through the air generate turbulence in their wakes, and this turbulence can persist for nearly 1–2 min or even longer after the aircraft has passed. Previous experiences have shown that this turbulence from an aircraft can pose a hazard to encountering aircraft. The magnitude of the wake vortex depends on the weight of the aircraft generating it and operation type of the leading and trailing aircraft. The Federal Aviation Administration (FAA) safety regulations enforce minimum separation times between aircraft to prevent collision and ensure air traffic flows safely. The minimum separation times enforced by FAA depend upon the operation type (land/take-off) and the size of the leading and the trailing aircraft. These minimum separation times are presented in the Table 9.1.

As Table 9.1 shows, minimum separation times are asymmetric, where the sequence of operations determines the actual separation times. Therefore, generating efficient aircraft schedules by exploiting the asymmetric separation times have a potential to increase runway utilization and delay reduction. However, the existence of asymmetric separation times between aircraft makes this scheduling problem much more difficult.

From an air controller's view the easiest and useable scheme for scheduling aircraft over runways is through the first-come-first-served (FCFS) order. For the landing aircraft this order will be based on the order they enter the radar range and

Table 9.1 Minimum separation times in seconds (FAA standard)

Departure → Departure				Departure → Arrival			
Lead/Trail	Heavy	Large	Small	Lead/Trail	Heavy	Large	Small
Heavy	90	120	120	Heavy	60	60	60
Large	60	60	60	Large	60	60	60
Small	60	60	60	Small	60	60	60
Arrival → Departure				Arrival → Arrival			
Heavy	75	75	75	Heavy	96	157	196
Large	75	75	75	Large	60	69	131
Small	75	75	75	Small	60	69	82

on the order of the aircraft queueing at the holding area for the taking-off aircraft. However, most of the time FCFS is not capable of providing the best schedule for runway utilization (Capr and Ignaccolo 2004).

The MRASP has been typically classified into two different categories: static (offline) and dynamic (online) case. The static case is solved before actual operations with known or predicted information, while the dynamic case is solved to generate schedules in real time. The most common researched model in the literature is the static (offline) case where all data including ready times, target times, and due times are assumed to be known beforehand and can be taken into account in the process. On the other hand, in the dynamic (online) case all these data become known only when an aircraft is ready to land or take-off. Unlike the static case, modeling approaches for dynamic case requires some additional considerations about which aircraft to reschedule and when to reschedule the sequence of the aircraft.

The MRASP can be viewed as an identical parallel machine scheduling problem in which "aircraft" and "runways" represent "jobs" and "machines," respectively. Classical parallel machine scheduling problem consists of assigning a number of jobs on a set of parallel machines with given release time, start time, and latest finish time for each job and sequence-dependent setup times that are based on the completed job and its succeeding job. The mapping of the identical parallel machine scheduling to MRASP relies on the following assumptions:

- If an aircraft begins to land or take-off, it cannot be interrupted by another aircraft.
- One aircraft is at most allowed to land on or take-off from each runway at any time.
- Runways are available and reliable at all times.
- Any aircraft can land on or take-off from at most one runway at any time.

In the literature the three term notation, $\alpha \mid \beta \mid \gamma$, is adopted, which is proposed by Graham et al. (1979) as the classification scheme for scheduling problems. In three term notation α indicates the machine environment, β describes the job and the resources characteristics, and γ defines the objective function to be minimized.

As a result, the MRASP is denoted by $P_m|s_{ij},tw|\Sigma w_j T_j$ where P_m denotes parallel machine scenario; s_{ij}, denotes sequence-dependent times between aircraft i and j, respectively, tw denotes time windows, and the objective is to minimize total weighted tardiness cost.

Different objectives are utilized in the literature considering different stakeholders' point of view, such as air traffic control system, airline companies, and airport management. Among these objectives the most commonly used ones are minimizing total tardiness (total delay), total earliness and tardiness (total deviation from the target time), and makespan (start time of the last aircraft). The total weighted tardiness is capable of addressing different stakeholders' needs by measuring the cost of delay that is a function of the length of delay multiplied by the weight (penalty) value related to each aircraft. Even the problem of single machine scheduling with total tardiness objective function is Non-deterministic Polynomial-time Hard (*NP*-Hard), i.e., it is unlikely that there can be developed a polynomial-time algorithm for finding an optimal schedule. The computational complexity of the identical parallel machine scheduling problem with total weighted tardiness objective function is therefore *NP*-Hard. Since exact algorithms require long computation times, different heuristics and metaheuristics are commonly employed to find near optimal values in shorter amounts of time. Therefore, this justifies the use of metaheuristics over exact methods for solving the MRASP.

9.2.1 Mathematical Programming Formulations

The literature presents two mathematical programming formulations for the problem. The first is a 0-1 MIP formulation and the second is a set partitioning formulation. Before presenting these formulations, the notation used throughout the chapter is shown below.

M: set of m independent runways, $M = \{1,2, \ldots, m\}$
N: set of n aircraft, $N = \{1,2, \ldots, n\}$
P: set of all feasible columns
i,j: aircraft indices
r: runway index
p: column (sequence of aircraft) index
r_j: ready time for aircraft j
δ_j: target time for aircraft j
d_j: due time for aircraft j
O_j: operation type of aircraft j
C_j: class of aircraft j
w_j: weight (penalty) value assigned to aircraft j based on its operation type and class
s_{ij}: Sequence-dependent separation time between aircraft i and j
a_j^p: 1 if aircraft j is covered by column p, 0 otherwise

Decision Variables:

t_j: start time of aircraft j
T_j: piecewise tardiness of aircraft j with respect to δ_j
z_{jr}: 1 if aircraft j is assigned to runway r, 0 otherwise
y_{ij}: 1 if aircraft i and j are assigned to the same runway and $t_j > t_i$ $(\forall i, j \in N,\ i \neq j)$, 0 otherwise
x_p: 1 if column p is involved in the solution, 0 otherwise

The MIP formulation that is given below for the MRASP is based on the formulation presented in Al-Salem et al. (2012) (the main difference from the MIP formulation presented by Beasley et al. (2000) is that two auxiliary binary variables are merged into one; one related to precedence on the same runway and the other related to whether aircraft pair are assigned to same runway.

$$min. \sum_{j \in N} w_j T_j \tag{9.1a}$$

$$s.t. \sum_{r \in M} z_{jr} = 1 \quad \forall j \in N \tag{9.1b}$$

$$1 \leq \sum_{j \in N} z_{jr} \leq \left\lceil \frac{n}{m} \right\rceil \quad \forall r \in M \tag{9.1c}$$

$$r_j \leq t_j \leq d_j \quad \forall j \in N \tag{9.1d}$$

$$t_j \geq t_i + s_{ij} - \left(1 - y_{ij}\right)\left(d_i - r_j + s_{ij}\right) \quad \forall i, j \in N,\ i \neq j \tag{9.1e}$$

$$y_{ij} + y_{ji} \geq z_{ir} + z_{jr} - 1 \quad \forall r \in M,\ \forall i, j \in N,\ i \neq j \tag{9.1f}$$

$$T_j \geq t_j - \delta_j \quad \forall j \in N \tag{9.1g}$$

$$0 \leq T_j \leq d_j - \delta_j \quad \forall j \in N \tag{9.1h}$$

$$z_{jr}, y_{ij} \in \{0, 1\} \quad \forall r \in M, \forall i, j \in N \tag{9.1i}$$

The objective function (Eq. 9.1a) is to minimize the total weighted tardiness. The constraints in Eq. (9.1b) ensure that each aircraft land on or take-off from exactly one runway. The constraints in Eq. (9.1c) are load-balancing constraints to enforce lower and upper bounds on the number of aircraft. The constraints (Eq. 9.1d) guarantee that each aircraft land or take-off within its time windows. The constraints (Eq. 9.1e) ensure the required separation times between any pair of aircraft. The constraints in Eq. (9.1f) actuate the sequencing variables between any pair of aircraft

that are assigned to the same runway. With the help of the constraints in Eq. (9.1f), the constraints in Eq. (9.1e) enforce separation only between aircraft that are assigned to the same runway. The constraints (Eq. 9.1g) specify aircraft tardiness, with respect to target times. The constraints in Eq. (9.1h) enforce non-negativity restrictions and upper bounds on aircraft tardiness. The constraints (Eq. 9.1i) define the binary decision variables.

The alternative mathematical programming formulation for MRASP is a set partitioning model. A set partitioning model aims to partition all elements into a number of subsets and each binary variable (column) represents a subset of elements defined by the coefficients. In MRASP, each column p represents a feasible sequence of aircraft with an aggregated cost. The set partitioning formulation, which is based on Ghoniem et al. (2015), is given below:

$$min. \sum_{p \in P} \left(\sum_{j \in N} w_j t_j a_j^p \right) x_p \tag{9.2a}$$

$$s.t. \sum_{p \in P} a_j^p x_p = 1 \quad \forall j \in N \tag{9.2b}$$

$$\sum_{p \in P} x_p = m \tag{9.2c}$$

$$x_p \in \{0, 1\} \tag{9.2d}$$

The objective function (Eq. 9.2a) minimizes the total weighted tardiness. The constraints in Eq. (9.2b), which are the set partitioning constraints, ensure that each aircraft is assigned to exactly one runway. The constraints in Eq. (9.2c) guarantee the limit on the number of the runways and constraints in Eq. (9.2d) are the integrality constraints on the decision variable x_p. The set partitioning problem is one of the first problems shown to be *NP*-Hard, therefore, no polynomial-time algorithm is likely to exist for this formulation either.

It is noteworthy to mention that each column p does not give the information related to the order of aircraft in that sequence. For large-scale problems it is computationally impractical to enumerate all columns.

9.3 Literature Review

Since 1960s, and especially in the past two decades, a considerable number and variety of research have dealt with aircraft scheduling on runways and a great deal of effort has been directed towards the development of solution algorithms for the problem.

Bennell et al. (2011, 2013) have provided a comprehensive review for airport runway scheduling. They indicated that several techniques have been formulated to solve runway scheduling problems and the main solution techniques include dynamic programming, branch-and-bound, heuristics, and metaheuristics. The solution methods for runway aircraft scheduling problems can be classified as: exact and heuristic algorithms. Exact algorithms, such as Branch-and-Bound, and dynamic programming guarantee optimal solutions, but they are extremely computationally intensive for large problem instances. On the other hand, heuristic algorithms generate solutions quickly but are not guaranteed to be close to the optimum and their performance is often evaluated empirically. Heuristic algorithms are usually classified as construction heuristics and improvement heuristics. Heuristic search methods, such as TS, Simulated Annealing (SA) are examples of the improvement heuristics.

Abela et al. (1993) studied the aircraft landing problem with a single runway and proposed a Branch-and-Bound algorithm based on a 0-1 MIP formulation. Ernst et al. (1999) also considered the single runway aircraft landing problem and pointed out that the single runway problem could be extended to multiple runways. They suggested a Branch-and-Bound algorithm and a genetic algorithm (GA). Beasley et al. (2000) addressed the landing problem, and proposed a MIP formulation and solved it by a technique based on the relaxation of binary variables by adding additional constraints. They also presented an effective heuristic algorithm.

Bianco et al. (2006) carried out a study examining the incorporation of a practical consideration which consists of constraining the set of feasible positions in the sequence for the new aircraft to prevent too much perturbation to the schedule. Artiouchine et al. (2008) proposed an approach based on a general hybrid Branch-and-Cut framework and used Constraint Programming and MIP, to solve the single runway problem with arbitrary time windows. Soomer and Franx (2008) studied a collaborative strategy where airlines assign a cost function for each of their flights and these cost functions are scaled per airline to achieve fairness among airlines. They also developed a local search heuristic to incorporate the fairness into the schedule.

Although the majority of the literature has focused on single runway scheduling problems, there are several published studies for MRASP. Ciesielski and Scerri (1998) suggested a GA for scheduling aircraft on two runways. Cheng et al. (1999) developed a GA for multiple runways. Wen et al. (2005) addressed the aircraft landing problem and formulated it as a set partitioning problem with side constraints. They suggested a Branch-and-Price algorithm, which is similar to the Branch-and-Bound, but column generation is applied at each node of the Branch-and-Bound tree. A combination of GA and an ant colony algorithm (ACO) for multiple runways has been proposed by Bencheikh et al. (2009). Pinol and Beasley (2006) suggested two population-based metaheuristics for multiple runways: scatter search and bionomic algorithm. Their objective was to achieve effective runway utilization, where two different objective functions (a non-linear and a linear) were used in the experiments. Hancerliogullari et al. (2013) proposed three greedy algorithms and two metaheuristics including SA and Meta-RaPS (Metaheuristic

for Randomized Priority Search). Liu (2011) presented a GA with a local search procedure incorporated into the GA framework, for solving the aircraft landing problem with runway dependent attributes. Xiao-rong et al. (2014) considered the multiple runways aircraft landing problem with the objective of minimizing the total deviation from the target time and suggested a hybrid bat algorithm, where several local search procedures are integrated into the framework.

A novel Branch-and-Price algorithm has been recently proposed by Ghoniem et al. (2015), who used the set partitioning formulation. The model decamped into master problem and pricing sub-problem and the pricing sub-problem was formulated as an elementary shortest path problem and solved with a specialized dynamic programming approach, which was identified as the main factor for accelerating the solution process substantially. Faye (2015) proposed a method based on an approximation of the separation time matrix by a rank 2 matrix and on discretization of the planning horizon. They suggested an exact method based on a dynamic constraint generation algorithm and also a heuristic method used to solve the model.

To the best of our knowledge, there is only one TS application for the aircraft scheduling problem. Atkin et al. (2007, 2008) dealt with the take-off scheduling with the objective to maximize the runway throughput. They proposed different metaheuristics (steeper descent, TS and SA) and analyzed their performance. TS outperformed the others but with a small margin.

Recently scheduling researchers and practitioners have been devoting more attention to the MRASP as can be noted from numerous recent publications However, little research has been undertaken to solve the problem in practical sizes in a timely manner. To the best of our knowledge, there is no work reported that deals with the problem of combined aircraft arrival-departure aircraft scheduling problem over multiple independent runways using a TS-based algorithm.

9.4 A Tabu Search Algorithm for the MRASP

TS is a single solution based search scheme proposed by Glover (1989, 1990) and has been applied successfully to solve many combinatorial optimization problems. TS is an iterative improvement algorithm based both on neighborhood search methods and the use of diverse types of memories and strategies to guide the search. The idiosyncratic characteristic of TS is its utilization of memory to guide the local search in order to escape from the local optimality. When a local optimum is faced, a move to the best neighbor is done even if this move may cause to worsen the objective function value. In order to avoid cycling a tabu list is utilized, which tracks attributes of recent moves and forbids any recurrence of such moves. Fundamental components of any basic TS algorithm are described below:

Search Space: Determining a search space along with a neighborhood structure is the most significant step of any TS implementation. The search space of TS is the space of all solutions that can be visited during the search. It is allowed to let the search move to infeasible solutions to escape local optimum.

Neighborhood Structures: Considering that the quality of the final solution relative to global optimum heavily depends on the structure of the neighborhood, a problem-specific neighborhood structure needs to be defined to cover all search space. There are several options for the neighborhood structures of the solution such as adjacent pairwise interchange, swapping, insertion, etc. Adjacent pairwise interchange requires exchanging positions of two elements directly next to each other. Swapping, or all pairwise interchange, entails exchanging positions of two different elements. Insertion is related to removing an element from its original position and placing it immediately after another one. Previously done computational experiments indicate that the insertion neighborhood structure produces better quality solutions than the swapping neighborhood structure (Laguna and Glover 1993). However, a hybrid neighborhood structure including both swapping and insertion has the potential to yield better solutions (Barnes and Laguna 1991).

Memory Structures: There are two types of memory, namely, explicit and attributive. Explicit memory is typically utilized for memorizing very good (elite) solutions encountered during the search. In contrast, attribute memory keeps the modifications that where done while proceeding from one solution to the next solution. Both explicit and attribute memories are used to build the short-term and the long-term memory of TS. For the short-term memory, a tabu list is retained in order to avoid cycling back to previously visited solutions. For the long-term memory, typically a frequency matrix is employed to detect more promising areas in the search space. It is important to note that short-term memory is used to store recency information, while long-term memory is used to store frequency information. The number of iteration that an attribute remains in the tabu list, which is referred to as tabu tenure, is also an important search parameter for TS. If the tabu tenure is too small, preventing the cycling might not be achieved; on the other hand, too long tabu tenure might create so many restrictions.

Aspiration Criteria: Due to the fact that a move or an attribute that is in the tabu list may forbid moving to attractive unvisited solutions, it is necessary to overrule the tabu status of this move or attribute in certain situations. This is achieved via the aspiration criteria. The most commonly used aspiration criterion is when the objective function value of the move under evaluation is better than the best objective function value found so far. In such case, the move will be taken even if it is on the tabu list.

Termination Criteria: The most commonly used termination criteria in TS are as follows:

- If the current iteration is equal to the maximum allowable iterations or the maximum allowable CPU time.
- If the current iteration is equal to the maximum allowable iterations without an improvement in the objective function value.
- If the objectives function value is equal to a pre-determined threshold value.

Main steps of a generic TS algorithm are given below. It is important to note that the term “solution” does not necessarily correspond to a final solution of the problem; it is just a solution in the search space.

Step 1: Generate all candidate solutions which are reachable by applying one move.
Step 2: Choose the best candidate solution that is not on the tabu list and the aspiration criteria if it is on the tabu list.
Step 3: Update the current solution and the best solution found so far.
Step 4: Determine if any termination criterion is satisfied. If yes, stop the algorithm; otherwise, go to step 2.

The performance of the basic version of TS explained above often needs to be improved to tackle difficult problems, because it tends to get stuck in a local optimum. In order to escape local optima, additional components for intensification and diversification need to be included in the search. Intensification is a myopic approach that is done by implementing some strategies to explore more thoroughly the promising areas of the search space. Diversification, on the other hand, is done by either performing several random restarts or implementing some strategies to penalize frequently performed move attributes. A rule of thumb approach for determining an intensification or diversification value is to analyze the moves that lead to elite solutions. This analysis can be done by decomposing these moves into moves attributes and deducing some special patterns. In general, move attributes that lead to good solutions contribute to higher intensification values and those that lead to worse solutions contribute to higher diversification values.

It is crucial to find a balance between the diversification ability to move towards new areas of the solution space and the intensification ability to explore intensely the most promising areas. Different metaheuristics utilize different strategies for controlling intensification and diversification. In TS it is usually done by controlling the length of the tabu list when fixed length tabu lists are used or by controlling the tabu tenure. The diversification effect will be stronger if the tabu list or the tabu tenure is longer, and the intensification effect will be stronger if the tabu list or tabu tenure is shorter.

We have designed a three-step algorithmic approach for the proposed TS algorithm to tackle the MRASP:

Step 1: Implement a Target Time First greedy algorithm (dispatching rule) to construct a feasible schedule.
Step 2: Apply the dispatching rule obtained in step 1 to an instance of the problem to generate an initial schedule.
Step 3: Input the initial solution and its objective function value to a TS algorithm to conduct an improving search until the termination conditions are satisfied. Lastly, return the best solution found during the search as the solution.

The basic structure of the proposed TS algorithm is, therefore, a variant of the tabu search. The overall TS algorithm is summarized in Algorithm 1, while the different components of the algorithm are explained in the remainder of the section.

Algorithm 1 Tabu Search Algorithm for the MRASP

Input: List of aircraft with time windows and separation times, and number of runways available for landing or take-off

begin
  Initialization
  find the initial solution s_0
  solution $s = s_0$, $s^* = s$, *tabuList* = empty
  while the termination criterion is not met
    update the iteration counter, $iter = iter + 1$;
    candidateList = empty, *bestCandidate* = null
    generate the set of candidate solutions (*candidateList*)
    for each candidate *sCandidate* in the *candidateList*
      if (*tabuList* does not contain *s***and** $f_{sCandidate} > f_{bestCandidate}$)
      than*bestCandidate* = *sCandidate*
    end for
    s = *bestCandidate*
    if ($f_{bestCandidate} > f^*$) **than** $s^* = bestCandidate$
    put *bestCandidate* in *tabuList*
  end while
end

Output: Best solution found so far, s^*, with an objective function value of f^*

9.4.1 Components of the Proposed TS Algorithm

Representation of a solution is encoded by m strings, where m is the number of runway. Each string contains aircraft land on or take-off from a runway and their sequences. The solution representation is complete because the set of aircraft sequences can represent all possible combinations. The objective function accumulates the penalty costs for all aircraft, which is in this case the weighted tardiness. The search space has to be carefully explored because there are two decision levels to consider: the assignment of aircraft to runways and sequencing aircraft in each runway.

A hybrid neighborhood generation scheme that consists of three types of moves is utilized: inter-runway swapping, intra-runway swapping, and insertion. Inter-runway swapping entails exchanging positions of two different aircraft on different runways and conversely intra-runway swapping on the same runway. Insertion is concerned with identifying two particular aircraft placed on different runways, placing one of them immediately after the other.

An attribute-based tabu list is employed and move attributes are recorded on the tabu list. Hence, tabu list prevents cycling and guides the search towards unexplored regions of the solution space. The aspiration criteria employed is to override the tabu restriction when the objective function value of the solution is better than any of the solutions found so far. Also, intermediate and long-term memory structures are

utilized to attain local intensification and global diversification. Intermediate term memory is activated by selecting the common attributes of some best candidate solutions during a specific period. The search procedure then tries to find new solutions that have these attributes. On the contrary, the long-term memory structure tries to force the search procedure to examine regions that are different from areas examined so far. To create diversification effect, it provides an evaluation criterion to generate new starting points. The termination criterion is met in our case if the best objective function value does not improve for a given number of iterations.

9.4.2 Initial Solution

Owing to the fact that beginning with a sufficiently good solution may improve the quality of the final result and amount of computational time needed, a dispatching (priority) rule is utilized as a greedy algorithm to find an initial solution for the problem. Dispatching rules are the most common form of constructive heuristics for machine scheduling problems based on which jobs awaiting processing on a machine are prioritizes. Whenever a machine is freed, a dispatching rule inspects the waiting jobs and selects the job with the highest priority. These rules have the potential to construct a reasonably good solution in a relatively short time and they are usually very easy to implement. Often a combination of basic dispatching rules can perform significantly better; however, there is large number of rules to choose from.

The solution obtained by Target Time First greedy algorithm is chosen as an initial solution. In this greedy algorithm, firstly, aircraft are ordered in ascending target time, and then assigned one by one to the runway with the least cost possible. The cost on a runway is calculated considering the previous aircraft and the corresponding separation time. Then an initial total cost is calculated according to the start time of each aircraft. Algorithm 2 below presents the pseudo code for this greedy rule.

9.5 Numerical Experiments and Results

Numerical experiments are carried out to evaluate the effectiveness of the proposed TS algorithm in terms of solution quality. Tests are conducted on a number of benchmark instances obtained from literature. The set of instances most often used for aircraft landing problems (Airland 1–13) are those in the Beasley's OR Library (Beasley 1990). However, these instances are considered easy for high performance computers and can be easily handled in reasonable time with state-of-the-art solvers. Therefore, the TS algorithm is tested against benchmark instances proposed in the literature by Ghoniem et al. (2015).

Algorithm 2 Target Time First Greedy Algorithm

Input: List of aircraft and number of runways, M

begin
  Initialization
  sort aircraft ordered in ascending target time (1 to N)
  for $i = 1$ to N
    for $r = 1$ to M
      calculate E_{ir} (Earliest feasible time that aircraft i can land on or take-off from runway r) from runway r)
    end for
    calculate start time $s_i = min\ \{E_{ir} \mid r \text{ in } M\}$
    assign aircraft i to the runway related to calculated s_i
  end for
  calculate the objective function $\left(\sum_{j \in N} w_j T_j\right)$
end

Output: A feasible solution consists of sequence of aircraft over runways with a fitness value and start time for each aircraft

In these benchmark instances, each aircraft is characterized by its ready time, target time, due time, operation type (arrival or departure), weight class (heavy, large, or small), priority (tardiness weight), and separation times with other aircraft. Every aircraft was set to a time window of 600 s. Ghoniem et al. (2015) provide a more detailed overview of these instances. These instances are composed of $N = \{15,\ 20,\ 25\}$ aircraft and $M = \{2,\ 3,\ 4,\ 5\}$ runways. A set of 55 different instances is proposed, at size $(N \times M)$ and they are denoted using the pair $(n,\ m)$ where n is the number of aircraft and m is the number of runways.

The proposed algorithm is implemented in a C++ environment on Microsoft Visual Studio 2013. All the experiments are performed on a standard PC machine with a 64 bit Intel(R) Core(TM) i5-3210M CPU 2.50 GHz processor and 8 GB of RAM running Microsoft Windows 8.1 operating system. A two-stage approach is utilized for the implementation. First the core data structures of the algorithm are created and then the algorithmic structure is built on them. Since TS heavily depends on memory structures, object-oriented techniques are employed sensible.

9.5.1 Parameters Setting

Although metaheuristics methods, including TS algorithm, are problem independent to a great extent, they still need some necessary parameter setting to adapt the method to the problem at hand and the choice of parameter values has a significant effect on the quality of the solution. Unfortunately there is no one-size-fits-all parameter setting for any given metaheuristic. For this reason, optimized values for

Table 9.2 Factors and their levels for TS algorithm

Factors	Low level (−1)	High level (+1)
A—Number of iterations after diversification is performed	15	25
B—Number of iterations after intensification is performed	10	15
C—Tabu tenure	7	12
D—Max number of successive iterations without improvement	50	100

Table 9.3 Regression statistics

Regression statistics	
Multiple R	0.9928
R^2	0.9875
Adjusted R^2	0.9367
Standard error	0.599
Observations	16

the parameters need to be determined carefully in a timely manner. Due to the fact that one-factor-at-a-time (OFAT) method does not consider the interactions between the parameters, which may significantly affect solution quality and time, a Design of Experiments (DoE) method, Taguchi design, is utilized to tune the parameters of the TS algorithm.

Due to the fact that determination of the factors and their initial levels require a priori knowledge of the behavior of the proposed algorithm on the problem instances, a pilot study is conducted. This study consists of several trials on a small subset of instances for preliminary analysis of the potential factors and their initial levels. From each aircraft-runway (n, m) configuration, one instance is selected randomly for these experiments. As a result of this pilot study, the TS factors that can influence the quality and computation time of the solutions with a low and high level for each factor is determined as a starting point. Therefore, the experimental ranges for each parameter are identified. The low and high levels are also denoted as −1 and +1, respectively, and are listed in Table 9.2.

After all the experiments are completed and the responses are calculated for each experiment, linear regression analysis is conducted to determine the significance of the parameters and their interactions. Linear regression analysis of the average relative error values produced a fit with R^2 value of 0.9875 (Table 9.3). The most significant factor is C, and the most significant interactions are BC and AC, respectively. This shows that the tabu tenure plays a significant role in the performance of the algorithm.

In order to verify the statistical validity of the results and to ensure that the effects of the different factor levels are statistically significant, the Main Effects Plot is used, where the mean values of each factor level are shown graphically in Fig. 9.1 in which we can see that C (tabu tenure) is the most significant factor. Also the Interaction Plots are used to determine the mean values for each factor level with the level of a

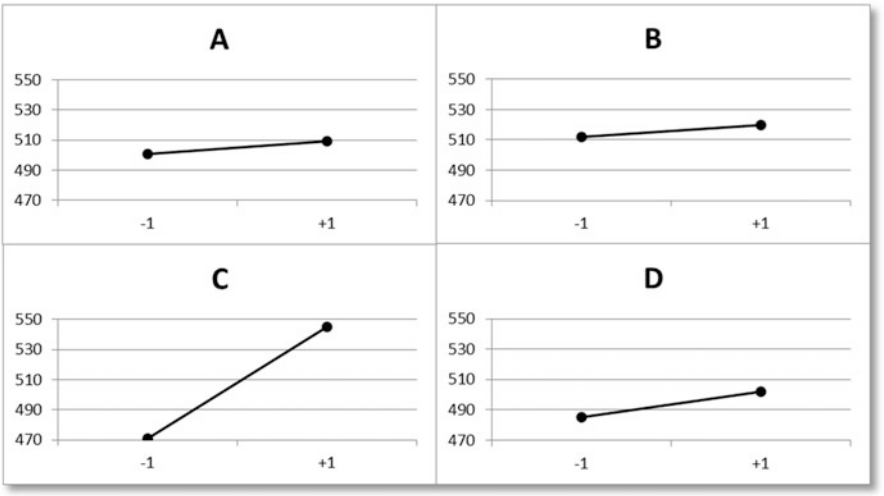


Fig. 9.1 Main effects plots

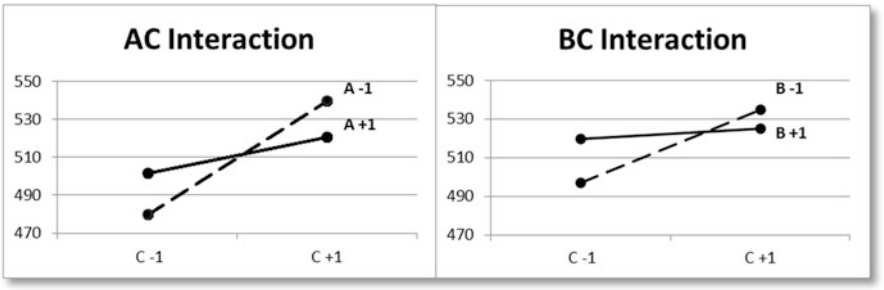


Fig. 9.2 Interaction plots

second factor held constant, which specifies that the effect of one factor is dependent on a second factor. Only the most significant second-order interactions (BC and AC) are presented in Fig. 9.2.

After finding a linear approximation of the response surface, the path of steepest descent on the response surface is calculated and small steps are made along this path by changing the parameter values. At each step, one trial is conducted and the process is continued until the limit of the experimental region is reached. The parameter vector associated with the best result found during this process is determined as the final parameter setting, where the most significant results are obtained with a medium levels for A and B, and high level for C. This implies that parameter values for the number of iterations after diversification and intensification is performed, and tabu tenure should be 22, 13, and 12, respectively.

9.5.2 Computational Results

The computational results for the benchmark instances are listed in Table 9.4. The objective function values of TS are reported in the columns represented by "Obj. Fn. Value." The percent deviation of TS values from the optimal solution ("Gap %") is calculated as:

$$Gap\ \% = \frac{(TS\ Obj.Fn.Value - Optimal\ Value)}{Optimal\ Value} \times 100 \tag{9.3}$$

The performance of the proposed TS metaheuristic algorithm is assessed based on the quality of the solution found via the gap between the optimal values and the objective function values. The results show that the solution found by the proposed Tabu Search algorithm has an average gap of approximately 10.15 %. Out of 55 instances, the algorithm found 9 optimal solutions. Also, the CPU times for the TS algorithm are listed showing that it is computationally efficient with computational time of less than half a second for any of the instances. The instances used in this experiment along with their solutions are available at www.SchedulingResearch.com.

In order to verify the statistical validity that hybrid neighborhood structure produce better results from the inter-runway swapping, intra-runway swapping, and insertion neighborhood structures applied alone, a small-scale experiment and an analysis of variance (ANOVA) is performed where the different neighborhood structures are considered as factors and the response variable being the average gap%. In this experiment one instance is selected for each (n, m) configuration and in total 11 instances are utilized. The results demonstrate that there is a clear statistically significant difference between performances of the neighborhood structures. The means plot and least significant difference (LSD) intervals (at the 95 % confidence level) for the neighborhood structures are presented in Fig. 9.3, which shows that the hybrid neighborhood structure works better than other neighborhood structures.

An experimental work is also carried out to determine the level of contribution of the intensification/diversification scheme to the overall algorithm. To be more precise, the purpose of the experiment is to find out whether the scheme used to determine the appropriate balance of diversification and intensification, does indeed provide the algorithm with the flexibility to deal with instances with different characteristics in an effective way. The 25 aircraft instances are utilized for this experiment due to the fact that these instances are more difficult and require a special intensification/diversification treatment. These instances are solved with and without intensification/diversification scheme. The algorithm is run for 1000 iterations for each instance and average gap% is collected for the algorithm with and without this scheme. Experimental results, as presented in Fig. 9.4, show that the proposed algorithm with intensification/diversification (I/D) scheme achieves better performance (faster convergence) over the one without this scheme as a consequence of its better exploration–exploitation capability.

Table 9.4 Computational results for the TS algorithm

(*n, m*)	Instance	Optimal value	Obj. Fn. value	Gap%	CPU times (s)	(*n, m*)	Instance	Optimal value	Obj. Fn. value	Gap%	CPU times (s)
(15, 2)	1	2139	2311	8.04	0.124	(20, 4)	29	521	586	12.48	0.486
(15, 2)	2	2173	2173	0.00	0.187	(20, 4)	30	587	1012	72.40	0.460
(15, 2)	3	6606	6631	0.38	0.137	(20, 5)	31	1645	1819	10.58	0.527
(15, 2)	4	3352	3372	0.60	0.157	(20, 5)	32	340	340	0.00	0.438
(15, 2)	5	1240	1240	0.00	0.133	(20, 5)	33	690	707	2.46	0.401
(15, 3)	6	581	627	7.92	0.214	(20, 5)	34	352	356	1.14	0.399
(15, 3)	7	1366	1381	1.10	0.302	(20, 5)	35	571	571	0.00	0.420
(15, 3)	8	761	962	26.41	0.466	(25, 2)	36	51	51	0.00	0.542
(15, 3)	9	2349	2855	21.54	0.413	(25, 2)	37	1801	1803	0.11	0.591
(15, 3)	10	753	787	4.52	0.215	(25, 2)	38	203	203	0.00	0.529
(15, 4)	11	2910	2952	1.44	0.275	(25, 2)	39	1232	1232	0.00	0.593
(15, 4)	12	3061	3107	1.50	0.237	(25, 2)	40	2798	2829	1.11	0.602
(15, 4)	13	2310	2480	7.36	0.342	(25, 3)	41	888	944	6.31	0.671
(15, 4)	14	452	655	44.91	0.530	(25, 3)	42	235	259	10.21	0.628
(15, 4)	15	468	508	8.55	0.471	(25, 3)	43	75	75	0.00	0.471
(20, 2)	16	5383	5495	2.08	0.341	(25, 3)	44	4442	4519	1.73	0.582
(20, 2)	17	2186	2947	34.81	0.576	(25, 3)	45	1602	1644	2.62	0.596
(20, 2)	18	669	669	0.00	0.357	(25, 4)	46	356	365	2.53	0.411
(20, 2)	19	1004	1043	3.88	0.278	(25, 4)	47	345	412	19.42	0.413
(20, 2)	20	1024	1068	4.30	0.392	(25, 4)	48	412	790	91.75	0.533
(20, 3)	21	471	475	0.85	0.423	(25, 4)	49	2084	2139	2.64	0.398
(20, 3)	22	1575	1616	2.60	0.378	(25, 4)	50	125	155	24.00	0.387
(20, 3)	23	1783	1839	3.14	0.423	(25, 5)	51	1094	1117	2.10	0.485
(20, 3)	24	1044	1115	6.80	0.570	(25, 5)	52	123	168	36.59	0.573
(20, 3)	25	2080	2098	0.87	0.340	(25, 5)	53	1578	1772	12.29	0.481
(20, 4)	26	354	361	1.98	0.354	(25, 5)	54	1379	1420	2.97	0.345
(20, 4)	27	632	657	3.96	0.356	(25, 5)	55	857	1196	39.56	0.544
(20, 4)	28	666	689	3.45	0.390	Average Gap %:				10.15	

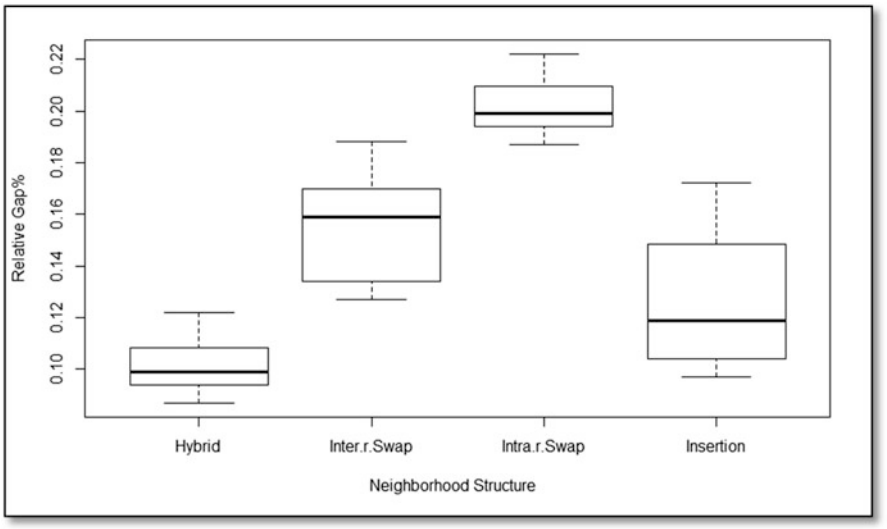


Fig. 9.3 Means plot and LSD intervals

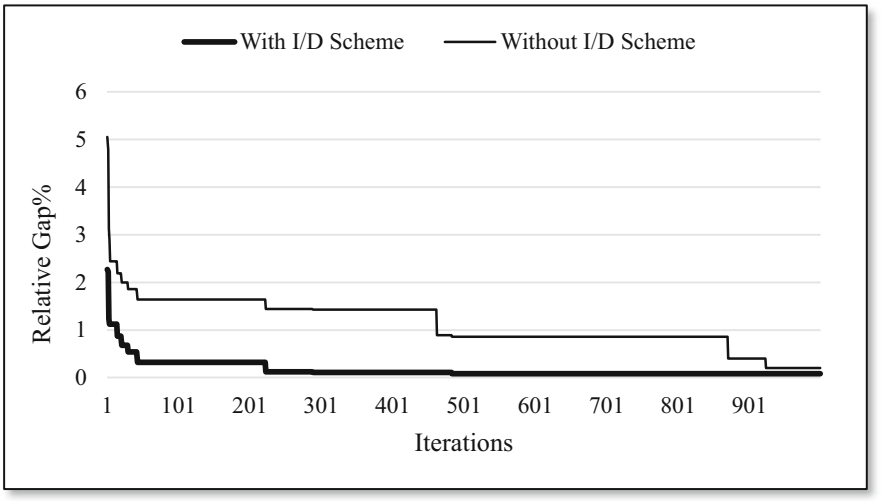


Fig. 9.4 Experimental results for I/D scheme

In order to test whether the greedy constructive heuristic algorithm improves the quality of the final result or not, we conducted an experiment in which we compared the algorithms with greedy initial solution against the one with a random initial solution. An instance for each (*n, m*) configuration is selected for the experiment and the average relative gap% is calculated for the comparison. As expected, the algorithm with greedy heuristic always produced a superior initial solution

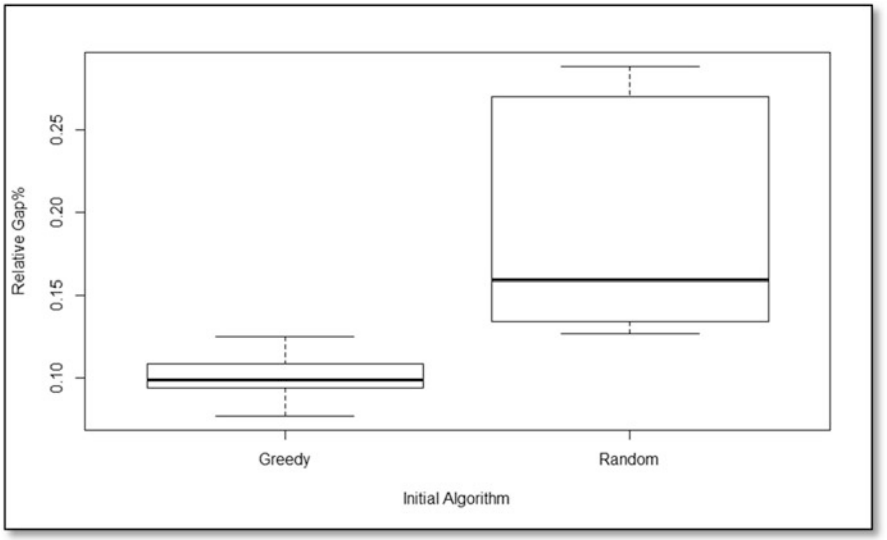


Fig. 9.5 Boxplot for comparison of initial solutions

compared to one with random initial solution, and as shown in Fig. 9.5, the final solutions obtained by the algorithm when the initial solution was produced by the greedy heuristic were better than its solution when the initial solution was random.

Based on the experimental results of the TS compared to the benchmark instances, it is effective and efficient. The success of the proposed TS algorithm may be credited to the following factors. First of all, the hybrid neighborhood structure made the search process very flexible. Second, the balanced intensification/diversification scheme improved the richness of the solution space. Finally, the greedy constructive heuristic algorithm for generating initial solutions was able to produce high quality solutions at the beginning of the process.

9.6 Conclusions

Considering that runway availability is the major limitation to airport capacity and adding more runways to increase the capacity is often not feasible due to a number of reasons including physical limitations and environmental restrictions, it is very important to utilize runways more efficiently. Therefore, developing methods applicable to MRASP is vital in terms of runway utilization and safety. Main challenges in solving MRASP include the pre-specified time windows and the asymmetrical sequence-dependent separation time requirements, which make the problem very challenging.

Some research exists for the MRASP especially for small-scale problems. However, real-life large-scale problems remain challenging from a computational standpoint. Heuristic or metaheuristic methods are often used to solve such problems, compromising optimality for computational time. In this chapter a Tabu Search (TS) based algorithm is introduced, where the memory forces the search process away from the local optimum. The proposed TS algorithm utilizes a hybrid neighborhood structure together with a balanced intensification/diversification scheme, which improved the richness of the solution space. Another strength of this TS is the greedy constructive heuristic to generate initial solutions. The proposed algorithm is tested on 55 benchmark instances from the literature. The experimental results, based on an experimental design, show that the proposed algorithm is able to solve problem instances in reasonable computing times with good solution quality.

Future research can explore different hybridization mechanisms such as path relinking and hybrid adaptive TS/set partitioning "matheuristic" which combines TS with classical mathematical programming approaches.

References

Abela J, Abramson D, Krishnamoorthy M, De Silva A, Mills G (1993) Computing optimal schedules for landing aircraft. Paper presented at the proceedings of the 12th national conference of the Australian Society for Operations Research, Adelaide

Al-Salem A, Farhadi F, Kharbeche M, Ghoniem A (2012) Multiple-runway aircraft sequencing problems using mixed-integer programming. In: Proceedings of the IIE annual conference, January 2012. Institute of Industrial Engineers, Chicago, p 1

Artiouchine K, Baptiste P, Dürr C (2008) Runway sequencing with holding patterns. Eur J Oper Res 189(3):1254–1266

Atkin JA, Burke EK, Greenwood JS, Reeson D (2007) Hybrid metaheuristics to aid runway scheduling at London Heathrow airport. Transp Sci 41(1):90–106

Atkin JA, Burke EK, Greenwood JS, Reeson D (2008) A metaheuristic approach to aircraft departure scheduling at London Heathrow airport computer-aided systems in public transport. Springer, Heidelberg, pp 235–252

Barnes J, Laguna M (1991) A review and synthesis of tabu search applications to production scheduling problems. Technical Report ORP91-05, University of Texas at Austin

Beasley JE (1990) OR-library: distributing test problems by electronic mail. J Oper Res Soc 41:1069–1072

Beasley JE, Krishnamoorthy M, Sharaiha YM, Abramson D (2000) Scheduling aircraft landings—the static case. Transp Sci 34(2):180–197

Bencheikh G, Boukachour J, Alaoui AEH, Khoukhi F (2009) Hybrid method for aircraft landing scheduling based on a job shop formulation. Int J Comput Sci Netw Secur 9(8):78–88

Bennell JA, Mesgarpour M, Potts CN (2011) Airport runway scheduling. 4OR 9(2):115–138; Ann Oper Res 204(1):249–270

Bianco L, Dell'Olmo P, Giordani S (2006) Scheduling models for air traffic control in terminal areas. J Sched 9(3):223–253

Capr S, Ignaccolo M (2004) Genetic algorithms for solving the aircraft-sequencing problem: the introduction of departures into the dynamic model. J Air Transp Manag 10(5):345–351

Cheng VHL, Crawford LS, Menon PK (1999) Air traffic control using genetic search techniques. In: Proceedings of the 1999 IEEE international conference on control applications, vol 1. IEEE, New York, pp 249–254

Ciesielski V, Scerri P (1998) Real time genetic scheduling of aircraft landing times. In: Evolutionary computation proceedings of the 1998 IEEE international conference on IEEE world congress on computational intelligence, May 1998. IEEE, New York, pp 360–364

Ernst AT, Krishnamoorthy M, Storer RH (1999) Heuristic and exact algorithms for scheduling aircraft landings. Networks 34(3):229–241

Faye A (2015) Solving the aircraft landing problem with time discretization approach. Eur J Oper Res 242(3):1028–1038. doi:10.1016/j.ejor.2014.10.064

Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, San Francisco, CA

Ghoniem A, Farhadi F, Reihaneh M (2015) An accelerated branch-and-price algorithm for multiple-runway aircraft sequencing problems. Eur J Oper Res 246:34–43

Glover F (1989) Tabu search-part I. ORSA J Comput 1(3):190–206

Glover F (1990) Tabu search—part II. ORSA J Comput 2(1):4–32

Graham RL, Lawler EL, Lenstra JK, Kan AR (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discret Math 5:287–326

Hancerliogullari G, Rabadi G, Al-Salem AH, Kharbeche M (2013) Greedy algorithms and metaheuristics for a multiple runway combined arrival-departure aircraft sequencing problem. J Air Transp Manag 32:39–48. doi:10.1016/j.jairtraman.2013.06.001

Laguna M, Glover F (1993) Integrating target analysis and tabu search for improved scheduling systems. Expert Syst Appl 6(3):287–297

Liu Y-H (2011) A genetic local search algorithm with a threshold accepting mechanism for solving the runway dependent aircraft landing problem. Optim Lett 5(2):229–245. doi:10.1007/s11590-010-0203-0

Pinol H, Beasley JE (2006) Scatter search and bionomic algorithms for the aircraft landing problem. Eur J Oper Res 171(2):439–462

Soomer MJ, Franx GJ (2008) Scheduling aircraft landings using airlines' preferences. Eur J Oper Res 190(1):277–291. doi:10.1016/j.ejor.2007.06.017

Wen M, Larsen J, Clausen J (2005) An exact algorithm for aircraft landing problem. IMM-Technical Report-2005-12, Informatics and Mathematical Modelling, Technical University of Denmark

Xiao-rong F, Xing-jie F, Rui Z (2014) Using the heuristic genetic algorithm in multi-runway aircraft landing scheduling. TELKOMNIKA Indones J Electr Eng 12(3):2203–2211

Chapter 10
Metaheuristic Approaches for Scheduling Jobs on Parallel Batch Processing Machines

Stefan Lausch and Lars Mönch

Abstract We consider a scheduling problem for parallel identical batch processing machines. A batch is a set of jobs that can be processed at the same time on a single machine. The jobs belong to incompatible job families. Only jobs of the same family can be batched together. We are interested in minimizing the total weighted tardiness (TWT) of the jobs. Problems of this type arise, for instance, in semiconductor manufacturing. Other known occurrence of batch processing machines can be found in gear manufacturing. We describe a genetic algorithm (GA), an ant colony optimization (ACO) approach, and a large neighborhood search (LNS) approach for this scheduling problem. The performance of the three metaheuristic approaches is compared based on randomly generated problem instances. The LNS scheme outperforms the two other metaheuristics and is comparable with a variable neighborhood search (VNS) approach, the best performing heuristic for this scheduling problem from the literature.

Keywords Parallel machines scheduling • Batch processing • Tardiness • Genetic algorithms • Ant colony optimization • Large neighborhood search

10.1 Introduction

Semiconductor manufacturing belongs to the most complex existing manufacturing processes. Several hundreds of jobs are processed on up to 100 different types of machines (cf. Mönch et al. 2011a) in semiconductor wafer fabrication facilities (wafer fabs). Up to one third of all operations in a wafer fab are performed on batch processing machines. A batch is a group of jobs that are processed at the same time on a single machine (cf. Mönch et al. 2013). Diffusion and oxidation

S. Lausch • L. Mönch (✉)
University of Hagen, Universitätsstraße 1, 58097 Hagen, Germany
e-mail: stlau@freenet.de; Lars.Moench@fernuni-hagen.de

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_10

operations in wafer fabs are typical examples for operations that are performed on batch processing machines, i.e., on furnaces. Because the processing times of jobs on batch processing machines are rather long compared to the processing times of non-batching machines, scheduling jobs on these machines has a large impact on the performance of the entire manufacturing system (cf. Mehta and Uzsoy 1998; Mathirajan and Sivakumar 2006; Mönch et al. 2013).

We consider a situation where we are interested in minimizing the TWT value of all jobs. It is assumed that all jobs are available for processing at time $t = 0$. Several metaheuristics are proposed for this problem (cf. Balasubramanian et al. 2004; Almeder and Mönch 2011). In the present chapter, we unify the corresponding algorithms by identifying major building blocks. In addition, we describe a new heuristic based on the principles of large neighborhood search (LNS). To the best of our knowledge, LNS-type approaches have not been used to solve batch scheduling problems.

This chapter is organized as follows. In the next section, we describe the scheduling problem. A corresponding mixed integer programming (MIP) formulation is provided. We then present several building blocks for metaheuristic-based approaches in Sect. 10.3. Three metaheuristics are described in Sect. 10.4. The results of computational experiments based on randomly generated problem instances are shown and analyzed in Sect. 10.5. Conclusions and some future research directions are presented in Sect. 10.6.

10.2 Problem Description

In this section, we start by describing the batch scheduling problem. We then present a MIP formulation for this problem.

10.2.1 Problem Formulation

We consider n jobs that have to be processed on m identical parallel machines. The jobs belong to F incompatible families. Only jobs of the same family can be processed together in a batch due to the chemical nature of the process. The maximum batch size on any of the machines is B. It is given in number of jobs. Job j has a weight w_j and a due date d_j. The family of job j is denoted by $f(j)$. All jobs of family f have the same processing time of p_f. We assume batch availability. The tardiness of job j is given by

$$T_j := (C_j - d_j)^+, \tag{10.1}$$

where C_j is the completion time of job j. In addition, we use the abbreviation $x^+ := \max(x, 0)$ in the rest of the chapter. The TWT value of the schedule is given by

$$\text{TWT} = \sum_{j=1}^{n} w_j T_j. \tag{10.2}$$

Using the $(\alpha|\beta|\gamma)$ notation from scheduling theory, the scheduling problem can be represented in the form:

$$P_m|p\text{-batch,incompatible}|\text{TWT}, \tag{10.3}$$

where P_m refers to m identical parallel machines and p-batch, incompatible to batching with incompatible job families. It is already shown by Mehta and Uzsoy (1998) that the single machine scheduling problem $1|p\text{-batch,incompatible}|\sum T_j$ is NP-hard. Hence, since this scheduling problem is a special case of Problem (10.3) listed above, the problem considered in this chapter, is also NP-hard. Therefore, we have to look for efficient heuristics to tackle large-scale problem instances in a reasonable amount of time.

A fairly simple example is depicted in Fig. 10.1. There are three furnaces and jobs that belong to the incompatible job families A, B, and C. The maximum batch size is $B = 4$ jobs. On Furnace 1, we have a full batch of family A, while full batches of family B are processed on Furnace 2 and 3, respectively. Several jobs of the three incompatible families are in the waiting room to be processed on one of the three furnaces.

There is one important property of a certain class of optimal schedules. It can be easily shown that there is at least one optimal schedule where batches are fully loaded, i.e., each batch contains B jobs, except maybe the last scheduled batch of each family that might contain less than B jobs. This result can be easily generalized to any regular criterion as shown by Mehta and Uzsoy (1998). Therefore, we can assume in the rest of the chapter that we know the number of batches to be formed. This number is denoted by n_b.

Three decisions have to be made to solve instances of Problem (10.3):

1. Form batches taking the maximum batch size and the incompatible families into account.
2. Assign the already formed batches to machines.
3. Sequence the set of batches for each single machine.

We will see in the remainder of this chapter that it is beneficial to make these decisions simultaneously.

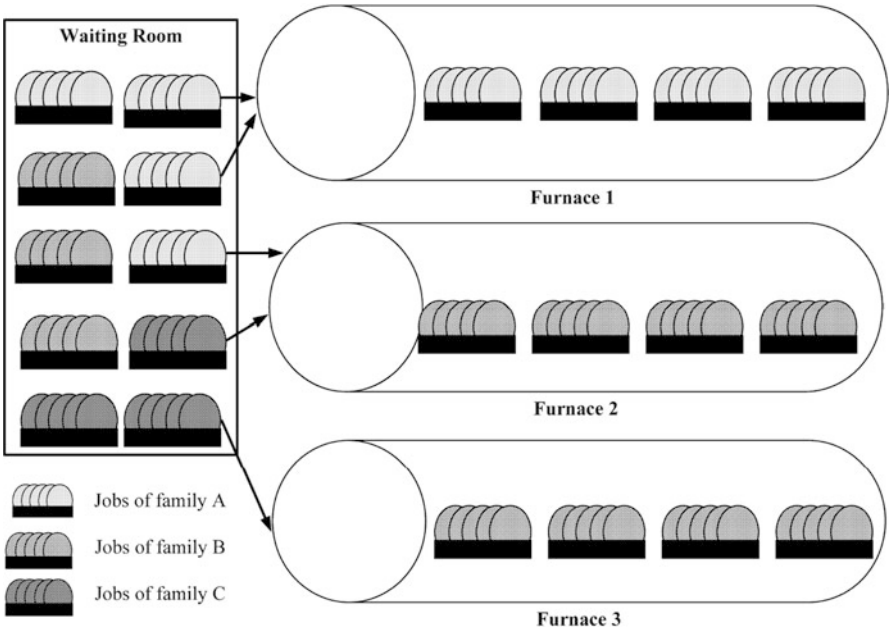


Fig. 10.1 Small-size example including three furnaces and three incompatible job families

10.2.2 MIP Formulation

Next, we present a MIP formulation for Problem (10.3) due to Almeder and Mönch (2011). We start by defining indices:

$b = 1, \ldots, n_b$: batch index
$f = 1, \ldots, F$: family index
$j = 1, \ldots, n$: job index
$k = 1, \ldots, m$: machine index.

The following parameters will be used within the model:

B : maximum batch size
d_j : due date of job j

$$e_{jf} : \begin{cases} 1, \text{ if job } j \text{ belongs to family } f \\ 0, \text{ otherwise} \end{cases}$$

M : large number
p_f : processing time of jobs that belong to family f
w_j : weight of job j.

The following decision variables are introduced:

C_{bk} : completion time of the bth batch on machine k

X_{jbk} : $\begin{cases} 1, \text{ if job } j \text{ is assigned to the } b\text{th batch on machine } k \\ 0, \text{ otherwise} \end{cases}$

Y_{bkf} : $\begin{cases} 1, \text{ if the } b\text{th of machine } k \text{ belongs to family } f \\ 0, \text{ otherwise} \end{cases}$

T_j : tardiness of job j.

Problem (10.3) may be formulated as follows:

$$\min \sum_{j=1}^{n} w_j T_j \tag{10.4}$$

subject to

$$\sum_{b=1}^{n_b} \sum_{k=1}^{m} X_{jbk} = 1, \quad j = 1, \ldots, n, \tag{10.5}$$

$$\sum_{j=1}^{n} X_{jbk} \leq B, \quad b = 1, \ldots, n_b, k = 1, \ldots, m, \tag{10.6}$$

$$\sum_{f=1}^{F} Y_{bkf} = 1, \quad b = 1, \ldots, n_b, k = 1, \ldots, m, \tag{10.7}$$

$$e_{jf} X_{jbk} \leq Y_{bkf}, \quad j = 1, \ldots, n, \ b = 1, \ldots, n_b, k = 1, \ldots, m, \tag{10.8}$$

$$p_f Y_{1kf} \leq C_{1k}, \quad f = 1, \ldots, F, k = 1, \ldots, m, \tag{10.9}$$

$$C_{b-1,k} + \sum_{f=1}^{F} p_f Y_{bkf} \leq C_{bk}, \quad b = 2, \ldots, n_b, k = 1, \ldots, m, \tag{10.10}$$

$$(C_{bk} - d_j) - M(1 - X_{jbk}) \leq T_j, j = 1, \ldots, n, \ b = 1, \ldots, n_b, \tag{10.11}$$

$$C_{bk}, T_j \geq 0, j = 1, \ldots, n, \ b = 1, \ldots, n_b, k = 1, \ldots, m, \tag{10.12}$$

$$X_{jbk}, Y_{bf} \in \{0, 1\}, j = 1, \ldots, n, \ b = 1, \ldots, n_b, \ f = 1, \ldots, F, k = 1, \ldots, m. \tag{10.13}$$

The objective (10.4) intends to minimize the TWT value. Constraints (10.5) ensure that each job is assigned to a batch. Constraints (10.6) do not allow more than B jobs to be assigned to the same batch. With constraints (10.7), we make sure that each batch belongs to a single job family, while constraints (10.8) ensure that the families of the jobs assigned to a batch match the family of the batch. Using constraints (10.9), the completion time of the first batch on a machine is computed, whereas constraints (10.10) ensure the correct completion times for all subsequent

batches. Finally, constraints (10.11) express the tardiness for each job. The non-negativity and binary constraints are modeled by expressions (10.12) and (10.13).

Note that the MIP formulation (10.4)–(10.13) is able to optimally solve problem instances up to 24 jobs, two families, and two machines within a reasonable amount of computing time using commercial solvers. Therefore, we can use the MIP to assess the solution quality of heuristics for small-size problem instances and to test whether the implementation of the heuristics is correct or not.

10.3 Building Blocks of Heuristic Approaches

We start by presenting a list scheduling approach to compute an initial solution in Sect. 10.3.1. We then present a decomposition heuristic to sequence already formed batches on a single machine in Sect. 10.3.2. In Sect. 10.3.3, a procedure that changes the content of already formed batches is discussed.

10.3.1 Determining an Initial Solution

We take a two-phase approach. Batches are formed in the first phase. These batches are sequenced in the second phase. We use the ATC index

$$I_j(t) := \frac{w_j}{p_{f(j)}} \exp\left\{-\frac{(d_j - p_{f(j)} - t)^+}{\kappa \bar{p}}\right\} \tag{10.14}$$

to sequence the jobs. Here, t is the time at which the machine becomes available, $\bar{p}$ is the average processing time of the remaining jobs, and κ is a scaling parameter. The jobs in each family are ordered in non-increasing order of their ATC indices. Then a batch of the first B jobs is formed for each family that contains unscheduled jobs. A batch is formed each time a machine becomes available.

When a batch B_k of family f is formed, it can be assessed using the BATC index. Therefore, we have

$$I_{\text{BATC}}(k, t) := \sum_{j \in B_k} I_j(t), \tag{10.15}$$

i.e., we sum up the ATC indices of the jobs that form the batch. The batch with the largest BATC index is scheduled. Therefore, BATC is a sequencing rule for already formed batches. It is well known that appropriate values for the look-ahead parameter κ lead to small TWT values (cf. Balasubramanian et al. 2004). Therefore, we use a grid search approach, i.e., we consider all values $\kappa \in \{0.5, 1.0, \ldots, 5.0\}$, fix one value from this set, solve Problem (10.3), and use finally the κ value that leads to the smallest TWT value for the given instance. The heuristic that forms

batches based on ATC and sequence the batches by BATC is called ATC-BATC-H in the remainder of this chapter.

10.3.2 Sequencing Batches on a Single Machine

Already existing sequences of batches can be improved by decomposition. This approach is used for single machine batch sequencing only and is similar to the one described by Mehta and Uzsoy (1998). A subsequence of batches of size λ is solved to an optimal sequence using complete enumeration. We have to consider $\lambda!$ different subsequences. The first α batches of this optimal subsequence are fixed into the final sequence, and the next λ unscheduled batches are considered. This process is repeated until all jobs are scheduled and no improvement is made in the TWT value of the final sequence. A maximum of *iter* iterations is allowed. The larger the value of λ, the higher the computation time and the better the solution quality. We call the heuristic based on the decomposition heuristic DH($\lambda, \alpha, iter$) in the remainder of this chapter.

10.3.3 Changing the Content of Already Formed Batches

Changing the content of batches is important to reduce the TWT value of schedules. To improve upon the solution obtained by ATC-BATC-H and DH, a corrective stage where the composition of batches is changed is included. After we get a complete solution of batches sequenced on all the machines, we attempt to improve the solution through the swapping techniques suggested by Devpura et al. (2000) for a single machine and extended by Balasubramanian et al. (2004) to the case of parallel machines.

The swapping algorithm is implemented as follows. We keep the position of the batches fixed and try to interchange jobs between batches of the same family. Initially for each family we consider the batch with the earliest start time. Considering all jobs in the batch starting from the first to the last, we look for the possibility of a swap with jobs in other compatible batches with start times later in the order of non-decreasing start times by calculating the TWT value of the old and the new schedule. Whenever a swap occurs we go back to the first job of the batch and start over. When there are no more improvements in this batch we move on to the next batch and check if we can swap the jobs in it with jobs in batches scheduled after it. The procedure is continued until all the batches of all the families are covered. We denote the swap procedure in the remainder of this chapter by Swap.

We will show in the next section how ATC-BATC-H, DH, and Swap will be incorporated in different metaheuristics.

10.4 Metaheuristic Approaches

We start by briefly recalling a genetic algorithm (GA) in Sect. 10.4.1. An ant colony optimization (ACO) scheme is presented in Sect. 10.4.2 following Almeder and Mönch (2011). A LNS scheme is described in Sect. 10.4.3.

10.4.1 Genetic Algorithm

A GA is a population-based metaheuristic that is used to solve hard combinatorial optimization problems. Various applications in operations management are known (cf. Aytuk et al. 2003). In this subsection, for the sake of completeness, we briefly recall the main features of the GA that is proposed by Balasubramanian et al. (2004). The GA is based on the idea that ATC-BATC-H is used to form batches. These batches are then assigned by the GA to the different machines. A batch-based representation is applied in the GA, i.e., we use chromosomes of the form:

$$c := (k_1, \ldots, k_{n_b}), \tag{10.16}$$

where $k_i \in \{1, \ldots, m\}$. Conventional one-point crossover and flip mutation are used as genetic operators. Finally, DH and Swap are applied to the best solution, i.e., the solution with the smallest TWT value computed by the GA. The algorithm can be summarized as follows:

Initialize: (1) Use the ATC-BATC-H procedure to form batches.
Algorithm: (2) Run the GA to assign the batches formed in Step (1) to the machines.
(3) Apply $DH(5, 2, 15)$ and Swap to improve the solution found in Step (2).

We see that all the three building blocks from Sect. 10.3 are required when the GA is used. Note that the GA can be easily changed into a memetic algorithm by applying DH and Swap to each chromosome. However, this approach will increase the computing time requirements of the algorithm.

10.4.2 Ant Colony Optimization

ACO is a population-based metaheuristic designed to solve hard combinatorial optimization problems. In a certain sense, it can be considered as a randomized list scheduling technique where learning takes place. For more details on ACO and its variants the reader is referred to Dorigo and Stützle (2004).

We start by describing the underlying principles of batch formation and assignment used within the ACO algorithm. We represent the scheduling problem (10.3) as a pure sequencing problem. Problem (10.3) is represented by a permutation

$$\pi := (\pi[1], \ldots, \pi[n]) \tag{10.17}$$

of the n jobs. At maximum B subsequent jobs of the same family form a batch. A final schedule for the batches can be obtained by assigning one batch after another to the next available machine starting with the first batch in (10.17). A permutation is constructed by adding the jobs of a newly formed and chosen batch to the end of the partial sequence starting from an empty sequence.

We consider an ordering of the jobs for each family f that are not included in the partial sequence of representation (10.17) to form batches:

$$\pi_f^t := (\pi_f[1], \ldots, \pi_f[n_{f,\text{curr}}]), \tag{10.18}$$

where $\pi_f[i]$ denotes the job of family f that is sequenced on position i in π_f^t and the number of jobs of family f that are not already part of the partial schedule that corresponds to (10.17) is $n_{f,\text{curr}}$. We determine π_f^t by sorting the jobs in non-increasing order with respect to a specific index. When we describe later a single iteration of the proposed ACO scheme, we will provide an ACO-specific index. The quantity t denotes the time when the newly formed batch can start with processing. This quantity is obtained by translating the partial sequence (10.17) into a partial solution of Problem (10.3) and by determining the time when the next machine is available. Starting from π_f^t, a sequence of batches is obtained by considering subsequences of the form:

$$B_{wf} := (\pi_f[wB + 1], \ldots, \pi_f[(w + 1)B]), \tag{10.19}$$

where $w \in \{0, \ldots, \lceil n_{f,\text{curr}}/B \rceil\}$. The batches of a family are sequenced in non-increasing order of importance, where each batch is evaluated by taking the sum of the indices that lead to π_f^t for all jobs of the batch. The used measure is described later, but for now its concrete form is not important. We select the most important batch among the batches $B_{0f}, f = 1, \ldots, F$ and add it to the partial sequence to construct the job permutation (10.17).

The necessary steps to obtain a solution for Problem (10.3) are:

1. Start with current time $t = 0$.
2. Form and order batches of the same family for all unscheduled jobs for starting time t.
3. Select a batch and add it to the end of the partial sequence (10.17).
4. Obtain the new starting time t of the next batch by interpreting the partial sequence (10.17) as a solution of Problem (10.3).
5. If there are still unscheduled jobs, continue with Step 2, otherwise stop.

Based on this procedure, it is enough to describe a heuristic to find job permutations that lead to small TWT values. An ACO-type approach is proposed to tackle this problem. The main idea of ACO is to let ants construct solutions of a given problem. This solution construction is guided by some memory, called pheromone, reflecting the solutions of previous iterations and using also heuristic information about the problem. While constructing the solution, ants provide information in those pheromone values to guide the search of the other ants, i.e., by performing a local pheromone update. The best ant updates the pheromone information to guide other ants of the next iteration, i.e., by carrying out a global pheromone update, after each iteration. In most applications of the ACO approach, solutions computed by ants are improved by applying a local search procedure. The overall scheme of an ACO algorithm (cf. den Besten et al. 2000) is as follows:

Initialize: (1) Set parameters.
(2) Initialize the pheromone trails.
Algorithm: (3) Repeat until termination condition is met

(a) Construct solution, perform step-by-step pheromone update.
(b) (Optional) Apply local search.
(c) Update pheromone trails globally.

In the remainder of this subsection, we describe the tailoring of the steps of the general ACO scheme to Problem (10.3). The heuristic information of the desirability for setting $\pi[i] := j$ is denoted by η_{ij}. The η_{ij} values are derived using the ATC dispatching rule with priority index (10.14) where t is the time when the next machine is available. We denote by $\tau_{ij}(l_{\text{iter}})$ the pheromone intensity that is associated with the setting $\pi[i] := j$, i.e., job j is placed on position i in π. The parameter l_{iter} is used for the current iteration of the ACO scheme. The maximum number of ants used within one iteration is denoted by $na_{\max}$.

The construction of a solution of an ant within one iteration of the ACO scheme works as follows:

1. Denote the set of jobs that are not already used to construct π by J_U. Initialize $J_U := \{1, \ldots, n\}$. We denote by J_U^f the following family-specific set:

$$J_U^f := \{j \in J_U | f(j) = f\}. \tag{10.20}$$

   Sequence the jobs $j \in J_U^f$ in non-increasing order with respect to the index

$$I_{ij}(l_{\text{iter}}) := \left(\sum_{l=1}^{i} \tau_{lj}(l_{\text{iter}})\right) \eta_{ij}^{\beta} \tag{10.21}$$

   to obtain π_f^t. Here, β is a parameter of the ACO approach.
2. Create a realization q_0 of an $U[0, 1]$-distributed random variable. We set $B_{f,\text{curr}} := \min(B, n_{f,\text{curr}})$ for $n_{f,\text{curr}} > 0$ in J_U^f. When $q_0 \leq q$ for a given q then the $\pi_f[1], \ldots, \pi_f[B]$ jobs that maximize the value of

$$\Phi_i^f := \frac{1}{B_{f,\text{curr}}} \sum_{g=1}^{B_{f,\text{curr}}} I_{i+g-1,\pi_f[g]}(l_{\text{iter}}) \tag{10.22}$$

over all families f are set into the current partial sequence on position $i, \ldots, i + B_{f,\text{curr}} - 1$. Using the sum of pheromone values of job j up to position i offers the advantage that the selection of jobs j that are chosen by previous ants for positions smaller than i will be enforced. The importance of the desirability is expressed by the parameter β. The parameter q is selected close to 1, i.e., a large portion of the new ants will find the next jobs to be scheduled with respect to the pheromone trail information and the heuristic information. Therefore, an ant makes the best decision with a probability of q as indicated by the heuristic information and the pheromone trails.

If $q_0 > q$ then job $j \in J_U^f$ is selected for position i according to the following discrete distribution with probabilities p_{ij} given by

$$p_{ij} := \begin{cases} \frac{(\sum_{l=1}^{i} \tau_{lj}(l_{\text{iter}}))\eta_{ij}^{\beta}}{\sum_{r \in J_U^f} (\sum_{l=1}^{i} \tau_{lr}(l_{\text{iter}}))\eta_{ir}^{\beta}}, & \text{if } j \in J_U^f \\ 0, & \text{otherwise} \end{cases}. \tag{10.23}$$

A job j is selected for each family according to the discrete probability distribution (10.23). We form the batch that includes this job and the next $B_{f,\text{curr}} - 1$ jobs from π_f^t.

Only the first job of the batch is selected according to the probability distribution (10.23) because we want to avoid completely destroying the structure of batches formed by ATC-type rules. The batches of all families are evaluated according to

$$\Phi_{ij}^f := \frac{1}{B_{f,\text{curr}}} \sum_{g=1}^{B_{f,\text{curr}}} I_{i+g-1,\pi_f[\pi_f^{-1}[j]+g-1]}(l_{\text{iter}}). \tag{10.24}$$

The batch with the largest Φ_{ij}^f value is considered. Therefore, the ants perform a biased exploration with a probability of $1 - q$. Notice that $\pi_f^{-1}[j]$ is the position of job j in the family-based ordering (10.18).

3. A local step-by-step update of the pheromone trail is performed immediately after an ant has added all jobs of a batch to the partial sequence by

$$\tau_{ij}(l_{\text{iter}}) := (1 - \theta)\tau_{ij}(l_{\text{iter}}) + \theta\tau_0, \tag{10.25}$$

where $\theta \in (0, 1]$ and $\tau_0 > 0$ are parameters of the ACO scheme. This pheromone update ensures that the decision to put job j on position i is less likely for consecutive ants. Therefore, exploring different sequences is ensured.

4. Remove all jobs added to the partial solution in the previous step from the set J_U. Repeat Step 2 if $J_U \neq \emptyset$. Improve the schedule by $DH(5, 2, 15)$ and Swap if $J_U = \emptyset$. DH and Swap work on the final schedule. Therefore, changes obtained by these procedures have to be incorporated in the representation (10.17).

   The pheromone values are updated after all ants of an iteration have constructed a solution. The global pheromone trail is updated by

$$\tau_{ij}(l_{\text{iter}} + 1) := (1 - \rho)\tau_{ij}(l_{\text{iter}}) + \frac{\rho}{\text{TWT}^*} \tag{10.26}$$

   if job j is scheduled on position i in the global best solution at iteration l_{iter}. Here, $\rho \in (0, 1]$ is a parameter that is used to model the pheromone evaporation. In addition, TWT^* is the smallest TWT value obtained so far by an ant.

The tailored ACO approach can be summarized as follows:

Initialize: (1) Choose $q, \rho, \theta, \beta, na_{\max}$.
(2) Use ATC-BATC-H to compute TWT_{best}. Set $\tau_0 := 1/\text{TWT}_{\text{best}}$.
Algorithm: (3) Repeat until termination condition is met

(a) Construct a solution based on ATC-BATC-H. Perform a step-by-step pheromone update according to expression (10.25).
(b) Apply $DH(5, 2, 15)$ and Swap.
(c) Update the pheromone trails based on expression (10.26) when the maximum number of ants is reached.

We see again that the three building blocks from Sect. 10.3 are used within the ACO scheme.

The proposed ACO scheme is again a population-based approach. However, we know from Almeder and Mönch (2011) that population-based approaches are outperformed by neighborhood search-based approaches when a fixed amount of computing time is given. This is our motivation to design a LNS approach for the problem at hand.

10.4.3 Large Neighborhood Search

LNS approaches are among the best performing heuristics for vehicle routing problems (VRPs) (cf., for instance, Ropke and Pisinger 2007 and Kovacs et al. 2012). Similar to VRPs, parallel machine scheduling problems are partition problems. Therefore, we expect an excellent performance of LNS-type algorithms for parallel machine scheduling problems.

There are only a few papers that deal with LNS approaches in machine scheduling. Flexible job shop problems are studied by Pacino and Van Hentenryck (2011) and Yuan and Xu (2013), while parallel machine scheduling problems are considered by Wang et al. (2012). However, the objectives and constraints in these papers are different from the ones used in Problem (10.3).

A set of destroy and repair methods compete in each iteration of a LNS scheme to improve the current solution. In this chapter, we use Adaptive LNS (ALNS) where multiple destroy and repair methods are used within the same search. Let x^b the best solution found during the search, while x is the current solution. We denote the corresponding TWT value by $TWT(x)$. The set of destroy and repair methods is denoted by Ω^- and Ω^+, respectively. Similar to Pisinger and Ropke (2010), a weight is assigned to each destroy and repair method that controls how often a specific method is applied during the search. This leads to vectors $\rho^- \in \mathbb{R}^{|\Omega^-|}$ and $\rho^+ \in \mathbb{R}^{|\Omega^+|}$ that are used to store the weights of the corresponding method. In the beginning, all methods have the same weight. The roulette wheel principle is used to choose a destroy and repair method in each iteration of ALNS. The probability to choose the destroy method $k \in \Omega^-$ is given by

$$\phi_k^- := \frac{\rho_k^-}{\sum_{i=1}^{|\Omega^-|} \rho_i^-}. \tag{10.27}$$

A similar expression is used for choosing the repair method l from Ω^+, namely

$$\phi_l^+ := \frac{\rho_l^+}{\sum_{i=1}^{|\Omega^+|} \rho_i^+}. \tag{10.28}$$

The components of the weight vector ρ are updated using score vectors $\psi^- \in \mathbb{R}^{|\Omega^-|}$ and $\psi^+ \in \mathbb{R}^{|\Omega^+|}$ for solutions. The overall scheme of an ALNS scheme is according to Pisinger and Ropke (2010) as follows:

Initialize: (1) Determine a feasible solution x of Problem (10.3).
(2) Set $x^b := x, \rho^- := (1, \ldots, 1), \rho^+ := (1, \ldots, 1), \psi^- := (0, \ldots, 0), \psi^+ := (0, \ldots, 0)$.

Algorithm: (3) Repeat until the termination condition is met

(a) Choose a pair (d, r) of destroy and repair methods $d \in \Omega^-$ and $r \in \Omega^+$ using the discrete probability distributions (10.27) and (10.28), respectively.
(b) Compute $x := r(d(x^b))$.
(c) Accept: If $TWT(x) < TWT(x^b)$ then $x^b := x$.
(c) Update the score vector and the vectors ρ^- and ρ^+.

Note that there are variants of ALNS where deteriorations of the current solution are accepted (cf. Pisinger and Ropke 2010). However, because these variants did not show superior performance in our computational experiments we do not present the corresponding algorithmic details due to space limitations. The update of the weight vectors ρ^- and ρ^+ is performed using the following updates of the components of the score vectors:

$$\psi_k^- := \begin{cases} \psi_k^- + \sigma_1, & \text{if the solution obtained in the current iteration is the} \\ & \text{global best one} \\ \psi_k^- + \sigma_2, & \text{if the solution obtained in the current iteration was not,} \\ & \text{visited before and improves the global best solution} \\ \psi_k^- & \text{, otherwise} \end{cases} \tag{10.29}$$

where σ_1 and σ_2 are parameters of the ALNS scheme. A similar update scheme is used for the components of ψ^+. The weights ρ_k^- are updated after every $i_{\max}$ iterations by the expression

$$\rho_k^- := \lambda \rho_k^- + (1 - \lambda)\psi_k^- / \max(1, \theta_k^-), \tag{10.30}$$

where $\lambda \in [0, 1]$ is a decay parameter controlling the impact of the weights and θ_k^- is the number of times the method k was used during the last segment of $i_{\max}$ ALNS iterations. The corresponding expression for the weights of the repair methods is given by:

$$\rho_l^+ := \lambda \rho_l^+ + (1 - \lambda)\psi_l^+ / \max(1, \theta_l^+), \tag{10.31}$$

where θ_l^+ is the number of times the method l was used during the last segment of $i_{\max}$ ALNS iterations. Weights that are unused in the current iteration do not change their value. On the one hand, large values for σ_1, σ_2 lead to a situation where a selection of the corresponding destroy and repair methods is more likely. The components of the score vectors $\psi^\pm$ are reset to 0 after the weights are updated.

We use the following destroy methods to tailor the ALNS scheme to Problem (10.3). Here, x denotes a feasible solution of Problem (10.3), whereas R is a number of batches to be removed from x.

1. *randBatchRem*(x, R): Randomly choose a machine. Randomly select a batch from this machine and remove it. This procedure is repeated until R batches are removed.
2. *randBatchPairRem*(x, R): This procedure is similar to *randBatchRem*(x, R). However, in contrast to *randBatchRem*(x, R) we remove batches from pairwise different machines, i.e., the machines considered in consecutive steps of the procedure are different.
3. *worstBatchRem*(x, R): Compute a list L where all batches from x in non-increasing order of their weighted tardiness values are included. Choose a random number $z \in [0, 1]$ and remove the batch $\tilde{b} := L[\lceil z^{p_{\text{worst}}} |L| \rceil]$ from the list and from x. Here, p_{worst} is a parameter of the ALNS scheme. The entire procedure is repeated until R batches are removed from x.
4. *relBatchRem*$(x, R, \delta, \phi, \mu)$: Let SB the set of all batches from x. Randomly select a seed batch b from x. Insert it into the list of removed batches RB. Sort all batches from $SB \setminus RB$ in non-decreasing order with respect to their relatedness to b where the relatedness of two batches b_i and b_j is defined by

$$R(b_i, b_j) := \delta |s_{b_i} - s_{b_j}| + \phi |P_{b_i} - P_{b_j}| + \mu |wT(b_i) - wT(b_j)|. \tag{10.32}$$

Here, s_b is the start time of batch b in x, P_b the position of batch b on one of the machines, and finally $wT(b) := \sum_{j \in b} w_j T_j$ the weighted tardiness of batch b. Let denote the resulting sorted list by L. The random number z is chosen from [0,1] and the batch $\tilde{b} := L[\lceil z^{p_{\text{rel}}} |L| \rceil]$ is removed from L and x and inserted in RB. Here, p_{rel} is a parameter of ALNS. Randomly select a new batch b^* from RB and repeat the entire procedure where b is replaced by b^* until $|RB| = R$. The related removal mechanism is proposed by Shaw (1998). The basic idea is that it is easier to interchange related batches.

5. *randAllMachRem*(x, R): The goal of this method consists in randomly removing batches from all machines. Therefore, we start by randomly selecting a machine from which no batch is removed so far. Randomly select a batch from this machine and remove it. This procedure is repeated until R batches are removed or one batch is removed from each machine.
6. *randBatchPosRem*(x, R): This method aims to remove batches that are on the same position on different machines. Therefore, we randomly choose a first machine. A batch is randomly selected from this machine and then removed. If there are batches on the same position on the remaining machines then remove these batches too until R is not reached. Repeat this procedure until R batches are removed from x.

The following repair methods are applied where x is a partial solution of Problem (10.3), and I is the set of batches to be inserted:

1. *randBatchIns*(x, I): Randomly select a batch from I. This batch is scheduled on a randomly selected position on a randomly selected machine. This procedure is repeated until all batches from I are scheduled. Apply DH and Swap to the resulting schedule.
2. *greedyBatchInsTWT*(x, I): We assume that the batches in I are sorted with respect to the sequence in which they are removed from the schedule. Select the first batch b from I and try to insert it on each possible position within x. Insert b on the position that leads to the smallest TWT increase. Set $I := I \setminus \{b\}$ and $x := x \cup \{b\}$. Repeat this procedure until $I = \emptyset$. Perform Swap on the resulting schedule.
3. *greedyBatchInsDiffMachines*(x, I): This repair method is similar to the repair method *greedyBatchInsTWT*(x, I). However, only positions on machines are tested that are different from the machine where the batch was scheduled before the destroy.
4. *greedyBatchParInsTWT*(x, I): Try to insert all batches from I on each possible position in x. Insert batch $b \in I$ into x that leads to the smallest possible TWT increase. Set $I := I \setminus \{b\}$ and $x := x \cup \{b\}$. Repeat this procedure until $I = \emptyset$. Perform the Swap procedure on the resulting schedule.

Note that we also test pure LNS schemes. However, ALNS typically outperforms LNS-type schemes. We see again that all the ingredients from Sect. 10.3 are used to design the ALNS scheme.

10.5 Computational Results

The design of experiments used in the computational experiments is presented first in Sect. 10.5.1. Implementation aspects and the parameter setting for the metaheuristics are described in Sect. 10.5.2. We then present and discuss the results of the computational experiments in Sect. 10.5.3.

10.5.1 Design of Experiments

We expect that the quality of the proposed metaheuristics depends on the number of jobs in the incompatible families, the number of families, the number of machines, and the due date setting scheme. Due dates are chosen as follows:

$$d_j \sim U\left[(1 - R/2)\mu, (1 + R/2)\mu\right], \tag{10.33}$$

where T is the percentage of tardy jobs and R the due date range. The quantity μ is calculated as follows:

$$\mu = \hat{C}_{\max}(1 - T). \tag{10.34}$$

Here, $\hat{C}_{\max}$ is an estimator for the makespan of a schedule that is given by $\hat{C}_{\max} = \frac{nE(p_f)}{mB}$.

The design of experiments is summarized in Table 10.1. In total, 1440 problem instances are considered. Note that these instances are already used in Almeder and Mönch (2011).

We are interested in comparing the TWT values of the three metaheuristics relative to the TWT values obtained by ATC-BATC-H. Therefore, we determine for the heuristic H the ratio

$$\frac{\text{TWT(ATC-BATC-H)-TWT(H)}}{\text{TWT(ATC-BATC-H)}}. \tag{10.35}$$

In addition, different amounts of computing time are used as in Almeder and Mönch (2011) to allow for a fair comparison of the metaheuristics. The maximum computing time per problem instance is 1 min. Such an amount of computing time is small enough to fulfill the requirements of real-world decision making in a wafer fab.

Table 10.1 Design of experiments

Factor	Level	Count
Number of families F	3,6,12	3
Number of jobs per family n_f/F	$180/F$, $240/F$, $300/F$	3
Maximum batch size B	4,8	2
Processing time p_f	2 with probability 0.2 4 with probability 0.2 10 with probability 0.3 16 with probability 0.2 20 with probability 0.1	1
Weight w_j	$\sim U[0, 1]$	1
Percentage of tardy jobs T	0.3, 0.6	2
Due data range R	0.5, 2.5	2
Number of machines m	3,4,5,6	4
Total factor combinations		288
Number of independent problem instances		5
Total number of problem instances		1440

10.5.2 Implementation Aspects and Parameter Setting

All the different heuristics are coded in the C++ programming language. The experiments for the GA and for the ACO scheme are carried out on a PC with Intel Pentium D processor with 3.2 GHz CPU and 4 GB of RAM. SUSE Linux 10 is used as operating system. The computational experiments for the ALNS scheme are conducted on a PC with Intel Core i5-4570 with 3.20 GHz CPU and 8 GB of RAM using 64 Bit Windows 7 as operating system.

The parameter settings for the GA and the ACO are the same as in Almeder and Mönch (2011). The same setting for ATC-BATC-H is taken within all experiments, i.e., we use $\lambda = 5, \alpha = 2$ and iter $= 15$. The parameter setting for ALNS is determined based on some preliminary computational experiments in connection with a trial and error strategy. We use $\lambda = 0.1$ as proposed in Kovacs et al. (2012), $i_{\max} = 100$, $p_{\text{rel}} = 2$, and $p_{\text{worst}} = 3$. In addition, the settings $\sigma_1 = 33$ and $\sigma_2 = 9$ are applied. For the related removal method, we use $\delta = 9$, $\phi = 3$, and $\mu = 4$. In each iteration of the ALNS approach, each destroy method removes between 5 and 12 % of the jobs. The concrete amount is randomly selected in each iteration. The corresponding number of batches to be removed is determined based on the number of jobs.

10.5.3 Results of the Computational Experiments

We compare the performance of the three different metaheuristics. In a first experiment, ALNS is applied to ten very small-size problem instances with up to 24 jobs that are solved to optimality using the commercial solver CPLEX. Similar to ACO, ALNS is able to determine an optimal solution within 5 s of computing time. This result indicates that ALNS is correctly implemented.

In a second experiment, we solve the 1440 problem instances that are described in Sect. 10.5.1. The corresponding results are shown in Table 10.2 where we present the values of the ratio (10.35) for the different metaheuristics. Three independent runs are performed for each problem instance and each amount of computing time. Instead of comparing all problem instances individually, the instances are grouped according to factor levels such as number of machines, maximum batch size, etc. For example, results for $m = 3$ imply that all the other factors have been varied, but the number of machines has been kept constant at 3. The results of the best performing algorithm within a specific row are marked in bold.

We see from Table 10.2 that ALNS clearly outperforms the GA and the ACO scheme. Even after a computing time of 5 s, ALNS is able to obtain better results than the GA and the ACO scheme after 1 min of computing time. The GA and the ACO scheme are population-based approaches. Because they have to deal with an entire population of solutions they show a more time-consuming behavior compared to ALNS and variable neighborhood search (VNS). The largest improvements can be observed for small values of T and R. In this situation, the due dates of the jobs are wide. As a result, ATC-BATC-H performs poorly. Hence, metaheuristics have much room for improvement since the sequencing decisions are reversible in contrast to list scheduling approaches.

The results obtained by ALNS are slightly better than those obtained by the VNS approach proposed by Almeder and Mönch (2011). This might be a result of the more powerful hardware used in the present experiments for ALNS. Therefore, we state that ALNS and VNS provide a comparable solution quality.

10.6 Conclusions and Future Research Directions

In this chapter, we discussed several metaheuristics for a batch scheduling problem with parallel identical machines. First, building blocks of metaheuristic approaches for the present scheduling problem were developed. The application of the different building blocks to construct heuristics was explained. A GA, ACO, and ALNS-based metaheuristic were described. We demonstrated by computational experiments that our ALNS scheme outperforms the GA and the ACO scheme and is comparable to the VNS scheme from Almeder and Mönch (2011) with respect to solution quality and computing time.

Table 10.2 Improvement of the three metaheuristics over ATC-BATC-H (in percent)

	GA				ACO				ALNS			
Compare	5 s	10 s	30 s	60 s	5 s	10 s	30 s	60 s	5 s	10 s	30 s	60 s
n												
180	3.12	4.99	7.00	7.34	6.55	6.87	7.19	7.37	**9.37**	**9.52**	**9.72**	**9.77**
240	1.36	3.22	6.31	7.18	6.64	7.07	7.52	7.74	**9.31**	**9.59**	**9.85**	**9.98**
300	−0.05	1.47	4.44	5.90	5.83	6.27	6.82	7.12	**8.08**	**8.47**	**8.94**	**9.14**
B												
4	0.20	1.79	4.77	6.11	6.23	6.65	7.08	7.31	**8.62**	**8.98**	**9.41**	**9.58**
8	2.62	4.53	6.94	7.39	6.33	6.70	7.16	7.39	**9.23**	**9.41**	**9.61**	**9.68**
m												
3	2.87	3.98	5.58	6.12	6.31	6.64	7.02	7.22	**8.21**	**8.43**	**8.71**	**8.81**
4	1.97	3.56	5.90	6.73	6.47	6.89	7.32	7.56	**8.71**	**8.97**	**9.29**	**9.43**
5	0.90	2.92	6.10	7.14	6.43	6.83	7.26	7.49	**9.21**	**9.50**	**9.79**	**9.95**
6	0.47	2.73	6.35	7.50	6.37	6.82	7.35	7.61	**9.57**	**9.88**	**10.24**	**10.33**
R, *T*												
0.5, 30 %	8.97	12.65	17.97	19.47	17.26	18.31	19.41	20.00	**23.17**	**23.69**	**24.16**	**24.38**
0.5, 60 %	1.57	2.40	3.75	4.23	3.35	3.56	3.81	3.96	**5.09**	**5.25**	**5.44**	**5.51**
2.5, 30 %	−2.21	−1.02	0.68	1.29	1.41	1.53	1.68	1.76	**2.71**	**2.80**	**2.95**	**3.00**
2.5, 60 %	−2.96	−1.65	0.75	1.75	2.81	3.03	3.30	3.41	**4.73**	**5.03**	**5.48**	**5.63**
F												
3	4.49	5.75	7.42	7.93	7.41	7.77	8.20	8.43	**9.32**	**9.49**	**9.66**	**9.72**
6	1.31	3.26	6.09	7.03	6.83	7.25	7.73	7.95	**9.41**	**9.70**	**10.01**	**10.17**
12	−1.34	0.69	4.26	5.49	4.79	5.20	5.63	5.87	**8.04**	**8.39**	**8.84**	**9.00**
Overall	1.34	3.10	5.79	6.78	6.21	6.61	7.05	7.28	**8.92**	**9.19**	**9.51**	**9.63**

There are several directions for future research. First of all it seems to be interesting to provide non-trivial lower bounds by designing column generation (CG) approaches for Problem (10.3). A corresponding CG scheme for a single machine batching problem without incompatible families and makespan criterion is already described by Parsa et al. (2010). However, to the best of our knowledge, the parallel batch processing machine case is not tackled so far by CG. We expect that the CG approach presented in Mönch et al. (2011b) can be extended to Problem (10.3). A second direction is given by designing problem-specific filtered beam-search heuristics. Some initial experiments for the scheduling problem $1|p\text{-batch,incompatible,non-identical}|\sum T_j$ (cf. Bücher 2014) indicate that beam-search algorithms can be used to compute high-quality solutions in a short amount of time. Here, we refer to non-identical job sizes by the notation non-identical. Hence, such techniques can be used to speed up the corresponding branch and bound approaches. Finally, it seems interesting to extend the ALNS scheme to the scheduling problem $P_m|r_j, p\text{-batch,incompatible}|\text{TWT}$, where we denote by r_j the release date of job j. Based on the results in Bilyk et al. (2014) we know that VNS performs well in this situation. Therefore, we expect a similar behavior for our ALNS scheme.

References

Almeder C, Mönch L (2011) Metaheuristics for scheduling jobs with incompatible families on parallel batch machines. J Oper Res Soc 62:2083–2096

Aytuk H, Khouja M, Vergara FE (2003) Use of genetic algorithms to solve production and operations management problems: a review. Int J Prod Res 41(17):3955–4009

Balasubramanian H, Mönch L, Fowler JW, Pfund ME (2004) Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness. Int J Prod Res 42(8):1621–1638

Bilyk A, Mönch L, Almeder C (2014) Scheduling jobs with ready times and precedence constraints on parallel batch machines using metaheuristics. Comput Ind Eng 78:175–185

Bücher B (2014) Exakte und heuristische Ablaufplanungsverfahren für Jobs mit inkompatiblen Jobfamilien auf einer Batchmaschine. Bachelor thesis, University of Hagen, Department of Mathematics and Computer Science

den Besten M, Stützle T, Dorigo M (2000) Ant colony optimization for the total weighted tardiness problem. In: Proceedings 6th international conference parallel problem solving from nature (PPSN VI), pp 611–620

Devpura A, Fowler JW, Carlyle M, Perez I (2000) Minimizing total weighted tardiness on a single batch processing machine with incompatible job families. Proc Symp Oper Res 2000:366–371

Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Boston

Kovacs AA, Parragh SN, Doerner KF, Hartl RF (2012) Adaptive large neighborhood search for service technician routing and scheduling problems. J Sched 15(5):579–600

Mathirajan M, Sivakumar AI (2006) A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. Int J Adv Manuf Technol 29:990–1001

Mehta SV, Uzsoy R (1998) Minimizing total tardiness on a batch processing machine with incompatible job families. IIE Trans 30(2):165–178

Mönch L, Fowler JW, Dauzére-Pèrés S, Mason SJ, Rose O (2011a) Scheduling semiconductor manufacturing operations: problems, solution techniques, and future challenges. J Sched 14(6):583–595

Mönch L, Ziarnetzky T, Devpura A, Fowler JW (2011b) A genetic algorithm based column generation scheme for parallel machine scheduling with total weighted tardiness objective. In: Proceedings of the VIII metaheuristic international conference (MIC) 2011, pp 299–307

Mönch L, Fowler JW, Mason SJ (2013) Production planning and control for semiconductor wafer fabrication facilities: modeling, analysis, and systems. Springer, New York

Pacino D, Van Hentenryck P (2011) Large neighborhood search and adaptive randomized decompositions for flexible job shop scheduling. In: Proceedings of the twenty-second international joint conference on artificial intelligence, 1999–2003

Parsa NR, Karimi B, Kashan AH (2010) A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. Comput Oper Res 37(10): 1720–1730

Pisinger D, Ropke S (2010) Large neighborhood search. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics. Springer, Berlin, pp 399–420

Ropke S, Pisinger D (2007) A general heuristic for vehicle routing problems. Comput Oper Res 34(8):2403–2435

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Proceedings fourth international conference on principles and practice of constraint programming (CP-98)

Wang P, Reinelt G, Tan Y (2012) Self-adaptive large neighborhood search algorithm for parallel machine scheduling. J Syst Eng Electron 23(2):208–215

Yuan Y, Xu H (2013) An integrated search heuristic for large-scale flexible job shop scheduling problems. Comput Oper Res 40(12):2864–2877

Chapter 11
Worm Optimization for the Traveling Salesman Problem

Jean-Paul Arnaout

Abstract In this research, a new metaheuristic called Worm Optimization (*WO*) is proposed, based on the foraging behaviors of *Caenorhabditis elegans (Worms)*. In particular, the algorithm will mimic the behaviors of worms including finding food, avoiding toxins, interchanging between solitary and social foraging styles, alternating between food exploiting and seeking, and entering a stasis stage. *WO* effectiveness is illustrated on the traveling salesman problem (TSP), a known NP-hard problem, and compared to well-known naturally inspired algorithms using existing TSP data. The computational results reflected the superiority of *WO* in all tested problems. Furthermore, this superiority improved as problem sizes increased, and *WO* attained the global optimal solution in all tested problems within a reasonable computational time.

Keywords Traveling salesman problem • Worm optimization • Metaheuristic

11.1 Introduction

In recent years, and as optimization problems are becoming more complex and exact solution approaches computationally infeasible, the arena of biologically inspired computing is becoming quite popular. The latter links several subfields together such as social behavior and emergence, relying heavily on the fields of biology, computer science, and mathematics with an objective of using computers to model living phenomena and concurrently analyzing life to improve the usage of computers (Bongard 2009). In particular, the discipline of swarm intelligence is receiving a great deal of attention from the operations research community as it has delivered good and efficient solutions to NP-hard problems that would not be possible using traditional optimization approaches. Swarm intelligence concentrates on the cooperative and collective behaviors resulting from the interactions of

J.-P. Arnaout (✉)
Gulf University for Science and Technology, Mishref, Kuwait
e-mail: arnaout.j@gust.edu.kw

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_11

individuals among themselves and with the environment, including examples such as colonies of ants, bees, and termites, flocks of birds, and schools of fish (Dorigo 2007). Following the same body of knowledge, we propose in this research a novel optimization algorithm called Worms Optimization (*WO*) that is based on the behaviors of *Caenorhabditis elegans* (informally known as "the worm").

C. elegans is a non-parasitic transparent roundworm that lives in moderate to pleasant soil environments (Wood 1988). According to Ferris (2013), it was initially named Rhabditides elegans by Maupas (1900), then placed in the subgenus *Caenorhabditis* by Osche in 1952, and finally raised to the genus status by Dougherty in 1955. Early research about *C. elegans* started by Brenner (1974) who investigated their genetics, and since then the worm has been extensively researched. In particular, this worm is the only organism to have its neuronal wiring diagram completed (Jabr 2012). It is important to note that *C. elegans* is a small, soil-dwelling nematode with only 302 neurons (Macosko et al. 2009). Nevertheless, these neurons allow *C. elegans* to achieve several intricate behaviors including finding food (Avery and You 2012), avoiding toxins (Xu and Deng 2012), interchanging between solitary and social foraging styles (Lockery 2009), alternating between "dwelling—food exploiting" and "roaming—food seeking" (Shtonda and Avery 2006), and entering a type of stasis stage (Hu 2007). Despite this, none of the findings from the *C. elegans*' research was used to develop a metaheuristic that is based on this worm's behaviors.

Worm Optimization algorithm (*WO*) effectiveness is illustrated by solving the traveling salesman problem (TSP) that deals with finding the shortest closed tour after visiting all the cities once and only once in a given set. The main difficulty in solving TSP lies in the great number of possible tours $(n-1)!/2$ for n cities (Larranaga et al. 1999). Consequently, the TSP is known to be NP-hard; i.e., no algorithm can solve all instances, especially large ones, in a practical computational time. As a result, the literature presents lots of heuristics and metaheuristics that were developed to solve TSPs approximately. As the TSP is one of the well-known and popular problems in optimization, we will not present a detailed review of the problem. In particular, this study's aim is not about solving TSP; instead, it is about introducing *WO* and highlighting its potential by testing it on a popular optimization problem, the TSP. For further understanding on TSP, the reader can refer to Whitley et al. (1989), Lin et al. (1993), and Basu (2012).

Some preliminary promising results for this problem have been reported by Arnaout (2014). All instances and solutions for the problem addressed in this chapter are available at SchedulingResearch (2015). The rest of this chapter is organized as follows. In Sect. 11.2, we explain the key behaviors of worms. In Sect. 11.3, the Worms Optimization algorithm is introduced. The computational tests are presented in Sect. 11.4 and finally we conclude this research in Sect. 11.5.

11.2 *C. elegans*: Behaviors and Characteristics

In this section, we describe the key behaviors of *C. elegans* along with the associated neurons. In particular, we will discuss feeding, toxins avoidance, foraging styles, dwelling and roaming, and dauer arrest.

11.2.1 Feeding

Shtonda and Avery (2006) showed that worms are able to distinguish food based on its quality and will constantly seek out higher quality food. In particular, the authors did the following test as described by Avery and You (2012): Hundreds of worms were placed on plates that contained two different types of bacteria (food), and at later times, the distribution of worms indicated that more worms are found in the better food. In other words, when a worm is exposed to different types of bacteria, it will select always the better source.

Hence, the worm when modeled will follow a greedy rule in selecting solutions, as it always moves to the better solution without considering the global bests.

11.2.2 Foraging Styles

The key foraging styles of *C. elegans* can be summarized by either solitary or social (Lockery 2009). Furthermore, one of the worm's neurons is the RMG inter/motor neuron, which controls aggregation and related behaviors (Macosko et al. 2009). In particular, in their experiments, Macosko et al. (2009) manipulated the RMG activities, and clearly showed that high RMG activity indicates a social behavior, while a solitary behavior is observed with low RMG activity. Additionally, social strains are attracted by pheromone secreted by other worms as well as good quality food, while solitary strains are repulsed by this pheromone and dispersed evenly across all types of food. These two foraging styles are depicted in Fig. 11.1.

Depending on the RMG level, a worm could be either "social, attracted to pheromone, aggregated, and drawn to good solutions (greedy rule)" or "solitary, repulsed by pheromone, and evenly dispersed across all solutions."

11.2.3 Toxins Avoidance

Zhang et al. (2005) showed that worms can learn to avoid odors associated with infection (bad food) through sensory neurons referred to as ADF. Once a worm gets infected, it will learn from this infection in the sense to avoid eating the same

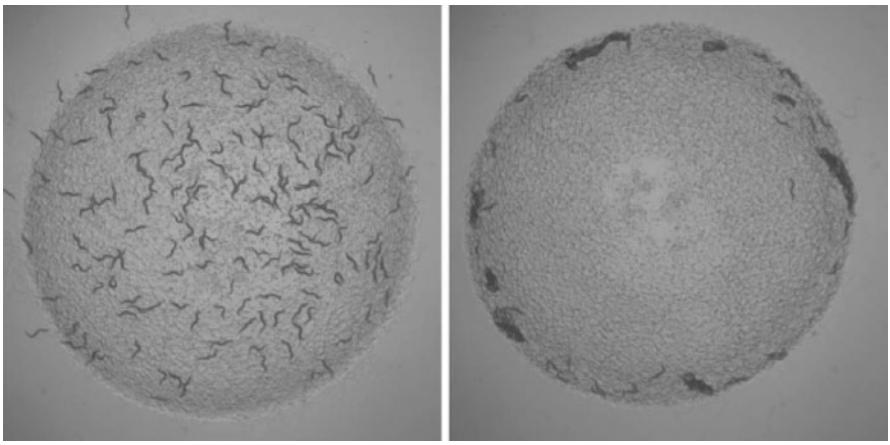


Fig. 11.1 This picture is taken from Macosko et al. (2009). The solitary strains are depicted in the *left figure*, which shows how they are evenly dispersed. The *right figure* highlights the aggregation of social strains towards good solutions

bad food again. The authors noted in their study that the avoidance of bad bacteria is similar to conditioned taste aversion, which is a known learning behavior in mammals, snails, and fish where the animals avoid food flavors that were related to intestinal distress.

The artificial worm will be equipped with a tabu-like list that stores a certain number of bad solutions, as not to visit them again for a certain number of iterations.

11.2.4 Dwelling and Roaming

Shtonda and Avery (2006) reported that an extrapharyngeal interneuron called AIY is critical for a worm to alternate between dwelling (local search) and roaming (global search). In particular, at lower AIY activity levels, the worm switched from "roaming—food seeking" to local search; the authors stated that this could be due to the worms perceiving the food to be of higher quality than it was.

Depending on the AIY level, a worm could switch to local search. Furthermore, if the food is of low quality, the worm will leave; i.e., will stop the local search.

11.2.5 Dauer Arrest

In harsh conditions, a worm enters a Dauer stage, where the latter describes an alternative developmental stage by going from a reproductive stage into a type of stasis and eventually dying if the environment conditions remain unfavorable.

The impacting conditions are many, but the prevalent ones are quality of food and increase in number/concentration of worms. In particular, if the present food quality is low and the concentration level is high; i.e., too many worms for low levels of food, then the worm enters dauer arrest. On the other hand, if the concentration level is low and the food supply is high, then the worms will reproduce.

Depending on the dauer status, the worm will move from a reproductive stage to a declining one, which would end once no more worms exist.

11.3 Worm Optimization

In this section, we translate the worm's behaviors discussed in Sect. 11.2 to the Worm Optimization algorithm and illustrate it using the TSP. First, it is worth indicating here that similarly to other swarm intelligence algorithms, and in order to solve an optimization problem using *WO*, the previous must be represented as a connected graph (nodes and arcs). The worm will move from one node to another in order to create a solution. The parameters needed in *WO* are as shown in Table 11.1. The actual values of the parameters (*if not constant*) are determined using Design of Experiments as shown later in Sect. 11.4.

11.3.1 Feeding Modeling

Initially, a certain amount of pheromone (τ_{ij}) is deposited in each arc in the network. The worms move from a node to another according to a probability, which is partially analogous to the one found in ant colony optimization (Dorigo and Gambardella 1997), with the addition of worms specialized attributes. The probability is determined by three factors: pheromone amount (τ), visibility (η), and bad solution factor (ADF). The probability of moving from node i to j for worm k can be calculated as shown in (11.1):

$$P_{ij}^{k} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}\mathrm{ADF}_{ij}}{\sum_{l\in\Psi}\tau_{il}^{\alpha}\eta_{il}^{\beta}\mathrm{ADF}_{ij}}, \tag{11.1}$$

where Ψ represents the set of unvisited nodes and the visibility (η) is determined depending if it is a social or solitary worm (explained in detail in Sect. 11.3.3). Two important parameters to direct the search are α and β, which are the exponents in the probability function that determine the importance of the pheromone amount over the visibility amount. Subsequently, when a worm finishes its tour; i.e., already visited all cities (nodes) and reported a solution, if this solution is better than the best solution found so far (*BestCost*), then the worm's tour (associated arcs) pheromone is updated according to Eq. (11.2).

$$\tau_{ij} \leftarrow \tau_{ij} \times (1+\rho) \qquad \text{if arc } (i,j) \text{ is used by worm } k \tag{11.2}$$

Table 11.1 Parameters of *WO*

Parameter	Range/Value	Description
Worm	(3, 60)	Worm number: number of worms in the algorithm (at the initialization stage)
MaxWorm	(5000, 15,000)	During the reproductive stage, the number of worms cannot exceed *MaxWorm*
RMG	(0.4, 0.7)	Lever between social and solitary strains: RMG = 1 indicates only social behavior and RMG = 0 only solitary. The range is not below 0.4 or higher than 0.7 as to not exclude a particular behavior
AIY	(0.1, 0.5)	Percentage of local search: the higher AIY, the higher the chance of dwelling (local search)
AIYIter	(5, 60)	Local search iterations: indicating the max number of local search iterations
BestIter	(100, 1000)	Number of solutions without improvement before concluding that food quality is bad
τ_{ij}	0.01	Pheromone: initial amount of pheromone deposited on arc (i, j)
ϕ	(0.1, 0.4)	Production rate: rate of reproduction of worms when they are not in Dauer stage
λ	(0.01, 0.5)	Bad solution factor: initially, the arc attractiveness (ADF) for all arcs equals 1; i.e., each arc has an equal probability of being selected. In the case of a bad solution, its associated arcs will be assigned an ADF = λ within the shown range in order to decrease its selection probability
ρ	0.01	Pheromone update: amount by which the pheromone is increased when a good solution is encountered
α	1.5	Exponent to determine the importance of the pheromone over the greedy rule
β	(0.5, 4.5)	Exponent to determine the importance of the greedy rule over the pheromone: if $\beta = 0.5$, pheromone is three times more important; if $\beta = 4.5$, greedy rule is three times more important
MaxIteration	(5000, 15,000)	Number of tours: referring to the total number of worms (tours) generated in *WO*

11.3.2 Foraging Styles Modeling

After *WO* initialization, and according to *RMG*, each worm is labeled as social or solitary. If the worm is social, it will be attracted to pheromone and will move according to the greedy rule; if the worm is solitary, it is repelled by the pheromone and will move with equal probabilities to possible cities. The pseudo code is summarized below:

Step 1: Generate random variable (rv) from U(0,1)
Step 2: If ($rv \leq$ RMG), then worm is social:

Step 2.1: for ($i = 1, \ldots, N$; $j = 1, \ldots, N$), $\eta_{ij} = 1/d(i,j)$,

Step 2.2: Equation (11.1) becomes: $P_{ij}^{k} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}\mathrm{ADF}_{ij}}{\sum_{l\in\Psi}\tau_{il}^{\alpha}\eta_{il}^{\beta}\mathrm{ADF}_{ij}}$.

Step 3: ElseIf($rv >$ RMG), then worm is solitary:

Step 3.1: for ($i = 1, \ldots, N$; $j = 1, \ldots, N$), $\eta_{ij} = 1/N$,

Step 3.2: Equation (11.1) becomes: $P_{ij}^{k} = \frac{1/\tau_{ij}^{\alpha}\eta_{ij}^{\beta}\mathrm{ADF}_{ij}}{\sum_{l\in\Psi}1/\tau_{il}^{\alpha}\eta_{il}^{\beta}\mathrm{ADF}_{ij}}$.

where $d(i,j)$ refers to the Euclidean distance between cities i and j.

In step 1, a random variable is generated to decide if the worm is solitary or social. If social, then in Step 2.1 the greedy algorithm assigns the closest city j to city i with a probability, and the higher the pheromone the more is the probability of this assignment. If solitary, then Step 3.1 suggests an equal probability $(1/N)$ of assignment for arc (i,j) and the higher the pheromone the less likely the assignment.

11.3.3 Toxins Avoidance: ADF Modeling

After a worm finishes its tour, its solution is assessed to determine if it should be added to the list of bad solutions or not. In this stage, we need to define an ADF list ($\mathrm{ADFList}_{h,N+1}$) to store the inferior solutions. This list can store up to h solutions, where $h = \left\lceil \sqrt{Worm} \right\rceil$; i.e., h is always changing depending on the dauer status (reproductive versus declining population). N is needed for every row to store the ordering of the N cities (nodes) in the solution and an additional cell to store the solution cost (*WormCost*). Furthermore, we define the array $\mathrm{ADF}_{N,N}$ to be linked with ADFList in the sense that for every solution stored in the list, its associated arcs are updated following $\mathrm{ADF}_{i,j} = \lambda$, where λ is the bad solution factor. The pseudo code for the ADF modeling, which is applied for every worm, is shown below:

Step 1: Sort *ADFList* in the ascending order according to *WormCost*
Step 2: Assess current worm's solution (*WormCost*)

Step 2.1: If ($WormCost > ADFList_{1,N+1}$):

– add worm's tour to *ADFList*,
– update the worm's associated arcs pheromone: $\tau_{ij} \leftarrow \tau_{ij}\times(1-\rho)$, *if arc* (i,j) *is used by worm k*

Step 2.2: ElseIf ($WormCost > ADFList_{h\ ,N+1}$), replace row h in the list with the current worm's tour,
Step 2.3: Else, $ADFList_{h\ ,i} = 0$, for $i = 1, \ldots, N+1$.

Step 3: for ($l = 1, \ldots, h$; $i = 1, \ldots N$),

Step 3.1: If ($ADFList_{l,i} <> 0$), then $\mathrm{ADF}_{l,i} = \lambda$.

Step 4: for ($i = 1, \ldots, N; j = 1, \ldots N$), Do: $\sum \text{ADF} = \sum \text{ADF} + \text{ADF}_{ij}$,

Step 4.1: If ($\sum \text{ADF} == N * \lambda$), then:
Step 4.1.1: choose a random city c,
Step 4.1.2:$\text{ADF}_{i,c} = 1$.

In Step 1, $\text{ADFList}_{1,N+1}$ stores the worst solution and correspondingly $\text{ADFList}_{h,N+1}$ stores the least bad solution. In Step 2, *ADFList* is updated according to *WormCost* quality. Note that in Step 2.1, if the solution generated was worse than the worst solution, the pheromone of the worm's tour is reduced. In Step 2.3, and in case the solution generated was not worse than the last tour stored in *ADFList* (*which represents the least bad solution*), then the latter is released from *ADFList* back into the search space. In Step 3, *ADF* values are updated based on the solutions that are present in *ADFList*. In Step 4, if $\sum \text{ADF} == N * \lambda$, this means that all candidate starting arcs in a tour have been added to the *ADFList*; we release a random arc in order to continue the search.

11.3.4 Dwelling and Roaming: AIY Modeling

When a worm generates a tour and its cost (*WormCost*), it has a probability of conducting local search. In this case, two cities from the worm solution are chosen at random and swapped. If this leads to a better solution, then the latter is retained; otherwise, the local search continues until either a better solution is found or the maximum iterations (*AIYIter*) is reached. In summary, after each worm completes its tour, the following is executed:

Step 1: Generate random variable (rv) from U(0,1).
Step 2: If ($rv \leq$ AIY), Do Local Search:

Step 2.1: for ($i = 1, \ldots,$ *AIYIter),* Do:
Step 2.1.1: choose 2 different cities (randomly) from the worm solution (*city1, city2*),
Step 2.1.2: swap *city1* and *city2* in the worm solution,
Step 2.1.3: if *WormCost* improved, then update solution; else, go to Step 2.1.

11.3.5 Dauer Arrest Modeling

Depending on the quality of the solutions obtained and worm concentration levels, a worm may enter Dauer. The food quality (*FQ*) is assessed after each worm finishes its tour. In particular, initially FQ = 0 indicating good food quality. In case the number of successive iterations without improvement exceeds *BestIter*, then FQ = 1; i.e., conclude that food quality is bad.

As for the concentration of worms in the search space, it is better assessed by the number of tours generated (i.e., worms); for each worm, the number of iterations is updated until it reaches its maximum (MaxIteration), which indicates high concentration of worms. Equation (11.3) is used to normalize the concentration to a (0,1) scale, where $WCt = 1$ indicates that *WO* reached the maximum number of iterations.

$$\text{Worms Concentration } (WCt) = \frac{\text{Iteration}}{\text{MaxIteration}} \tag{11.3}$$

Following each worm's tour, the Dauer status (*DauerStatus*) is assessed as follows:

Step 1: Compute *FQ* and *WCt*.
Step 2: Compute Dauer level: $DauerStatus = \frac{FQ+WCt}{2}$.
Step 3: Assess if worms are in reproductive or declining stage:

Step 3.1: If $(DauerStatus < 1)$, then $Worm = \min\{MaxWorm, \lceil Worm * (1+\phi)\rceil\}$,
Step 3.2: If $(DauerStatus == 1)$, then $Worm = \lfloor Worm * (1-\phi)\rfloor$.

Step 4: If $(Worm == 0)$, Stop *WO* and report *BestCost*.

11.3.6 Worm Optimization Modeling Summary

According to the above, Fig. 11.2 summarizes the steps of *WO* to solve the TSP. The pseudo code can be summarized as follows:

Step 1: Initialize *WO* Parameters and populate τ_{ij} and ADF_{ij} with the specified amounts.
Step 2: While $(Worm \neq 0)$, Do:

Step 2.1: $Iteration = Iteration + 1$;
Step 2.2: Solve for a tour according to Sect. 11.3.2 (Social versus Solitary)
Step 2.3: Find *WormCost* associated with Step 2.2
Step 2.4: Execute the local search approach (according to *AIY*) described in Sect. 11.3.4
Step 2.5: Update the pheromone according to Eq. (11.2)
Step 2.6: Execute the bad solutions approach described in Sect. 11.3.3 (*ADFList* and *ADF*)
Step 2.7: Update *Worm* number according to the Dauer approach in Sect. 11.3.5

Step 3: Output the best tour and its cost, Stop *WO*.

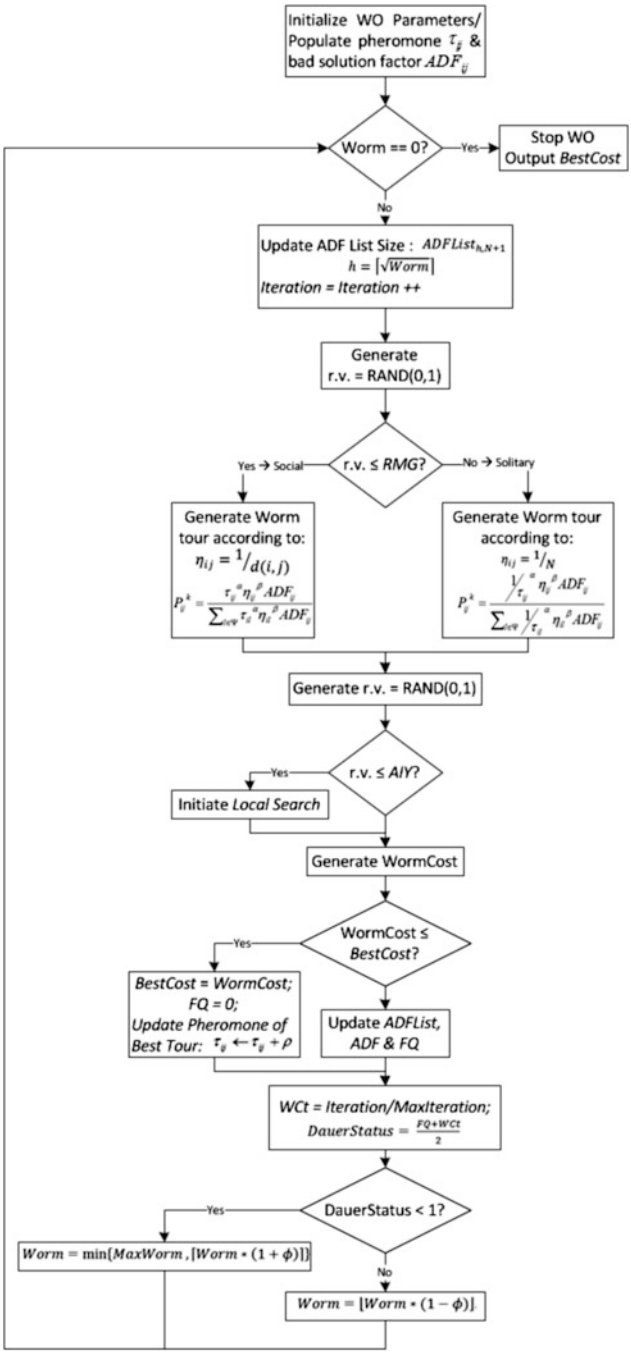


Fig. 11.2 Flowchart of the *WO* algorithm for TSP

11.4 Computational Tests

The proposed *WO* was coded in the C++ programming language running on Windows XP with a Pentium 4 processor. Design of Experiments (DoE) was utilized to determine the appropriate values for the *WO* parameters that will minimize the tour cost. Numerous publications provide a good review of DoE (e.g., Fisher 1960; Taguchi 1993; NIST/SEMATECH 2008).

The factors considered in this experiment with their levels of low and high are presented in Table 11.1. The values of the parameter levels were selected based on many runs under different settings. To reduce the number of runs but reach sound conclusions, D-Optimal design was utilized, which has been shown to be an effective design (NIST/SEMATECH 2008). JMP 10.0 from SAS was used to generate a D-Optimal design, with 112 experiments. The factors along with their interactions were analyzed using regression, ANOVA, and factors' effect tests. Three-factor interactions and higher were not considered as they typically have weak effect (Ross 1996). Based on a 95 % Confidence Interval, a relatively large t-Stat, and a small p-value (less than 0.05), the following parameter values were determined to provide the best performance for *WO*: Worm = 10, $MaxWorm = 6612$, RMG = 0.57, AIY = 0.36, $AIYIter = 44$, $\beta = 4.09$, $\phi = 0.3$, $\lambda = 0.01$, $BestIter = 125$, and MaxIteration = 8665, along with the predetermined parameters $\rho = 0.01$, $\alpha = 1.5$, $\tau_{ij} = 0.01$.

Figure 11.3 shows solution progress in terms of the *WormCost* in each iteration for a sample TSP to demonstrate the effectiveness of *WO* at each iteration in order to develop better solutions. This is manifested through the convergence of the *WormCost* after some running time, indicating that the search is not random.

In the preliminary tests that were conducted in Arnaout (2014), *WO* was compared to genetic algorithm (GA) and simulated annealing (SA) on few problem instances, and the results clearly indicated that SA performed the worst. To better test the performance of *WO*, it was compared to other naturally inspired optimization methods: ant colony system (ACS), particle swarm optimization (PSO), ant colony optimization with multiple ant clans (ACOMAC), as well as GA. All algorithms were tested on well-known TSPs that are included in TSLIB. In particular, the TSPs are Oli30 from Oliver et al. (1987), Eil51and Eil76 from Eilon (1969), KroA100 from Krolak et al. (1972), and d198 from Reinelt (1994). Results using ACS and ACOMAC are from Tsai et al. (2004), GA results are from Bersini et al. (1995) and Yang et al. (2008), and PSO are from Shi et al. (2007). The results are shown in Table 11.2, where the rows report the solutions obtained by the algorithms for the problem instances starting with the best known solution in the first row, and the "Type" column refers to the reported solution type depending if it is integer or real. Furthermore, as we compared against several algorithms and studies, we report "N/A" if the study did not test the related problem. Finally, we used two studies for the GA results and reported the best solution for each individual problem.

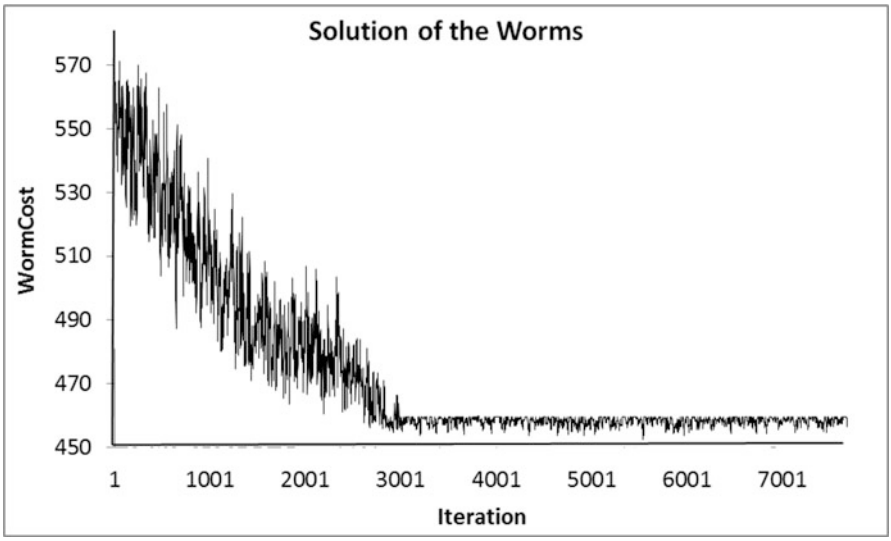


Fig. 11.3 WormCost vs *WO* iterations

It is clear from Table 11.2 that *WO* outperformed the other algorithms in all problem sizes and attained the optimal solution in all test problems. Nonetheless, and as in some problems PSO and ACOMAC's solutions were close to *WO*, the percentage deviation (δ) of the algorithms from *WO* was used as a measure of performance for each problem. That is:

$$\delta = \frac{\text{TourCost}_{\text{Algorithm}} - \text{TourCost}_{\text{WO}}}{\text{TourCost}_{\text{WO}}} \times 100\,\% \tag{11.4}$$

Figure 11.4 shows the values of δ for the real solutions which were reported only for ACS and ACOMAC. It can be seen that ACS performed the worst, followed by ACOMAC. Furthermore, both algorithms' deviation from *WO* increased with problem sizes (except in KroA100). Figure 11.5 shows the values of δ for the integer solutions using PSO and GA, where PSO performed better than the latter except for d198. Furthermore, it can be seen that PSO solution deteriorated with the problem size. Finally, in order to compare the performance of all algorithms, the average of δ was computed for every algorithm over its tested problems and depicted in Fig. 11.6. The latter shows that the four algorithms clearly deviated from *WO*, with ACOMAC performing the closest, followed by GA and PSO (*with GA performing slightly better*), and ACS performing the worst.

An important attribute of any metaheuristic is its computational time; unfortunately, the computational time of *WO* cannot be compared to the algorithms in this study as these came from different works and were run on different processors; i.e., such comparison would not be consistent across the algorithms. Instead, in this chapter, we will use the approach that was adopted by Dorigo and Gambardella

Table 11.2 Comparison of *WO* with *ACS, ACOMAC, PSO, and GA* with respect to total distances

Study Study	Method Method	Type Type	01i30 30 cities	EiI51 51 cities	Ei176 76 cities	KroA100 100 cities	d198 198 cities
Best known solution		*R*	423.74	429.9833	545.3876	21,285.44	15,809.66
		l	420	426	538	21,282	15,780
Proposed method	**WO**	*R*	**423.74**	**429.1179**	**545.3876**	**21,285.44**	**15,809.66**
		l	**420**	**426**	**538**	**21,282**	**15,780**
Tsai et al. (2004)	ACOMAC	*R*	N/A	430.684	555.23	21,457.93	16,041
		l		N/A	N/A	N/A	N/A
Shi et al. (2007)	PSO	*R*	N/A	N/A	N/A	N/A	N/A
		l		427	546	21,761	16,186
Bersini et al. (1995)[1]	GA	*R*	N/A	N/A	N/A	N/A	N/A
Yang et al. (2008)[2]		*l*	421[1]	430[2]	552[2]	21,761[1]	16,108[2]
Tsai et al. (2004)	ACS	*R*	N/A	434.178	559.7041	21,684.64	16,826.6
		l		N/A	N/A	N/A	N/A

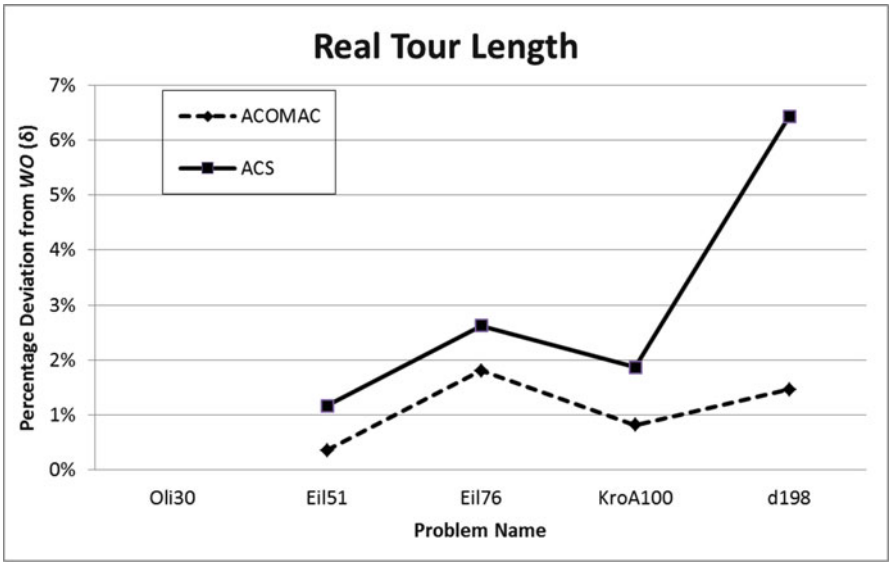


Fig. 11.4 Deviation of algorithms from *WO* (δ) —*Applicable to ACOMAC and ACS only (Table 11.2)*

(1997), where *WO* will be compared to the algorithms with respect to the number of tours required to find the best integer tour length. Since Dorigo and Gambardella's study did not cover PSO and ACOMAC, and given that our objective is to highlight *WO* computational performance, this part will compare between *WO*, ACS, and GA

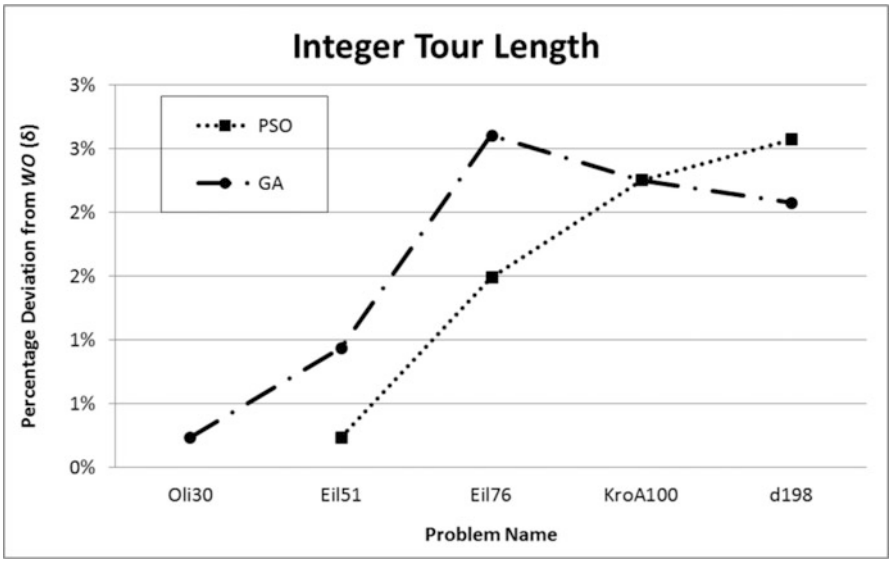


Fig. 11.5 Deviation of algorithms from *WO* (δ) —*Applicable to PSO and GA only (Table 11.2)*

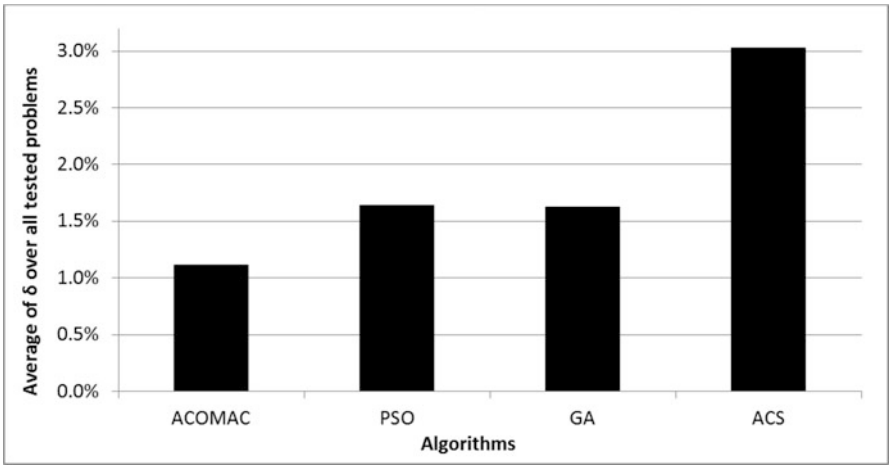


Fig. 11.6 Average of deviation (δ) for all problems (real and integer)

only. These computational results are shown in Table 11.3, where the performance of ACS and GA was obtained from Dorigo and Gambardella (1997).

It is clear from Table 11.3 that *WO* significantly outperformed GA in the number of tours required, especially as the problem size increased. This constitutes an affirmation of *WO*'s performance where the latter reached better solutions than GA in all problems while generating a significantly lower number of tours. On the other

Table 11.3 Computational comparison with respect to number of tours

Algorithm	# of tours required for best integer tour				
	Oli30	Eil50	Eil75	KroA100	d198
WO	842	1863	3567	4960	596,019
ACS	830	1830	3480	4820	585,000
GA	3200	25,000	80,000	103,000	N/A

hand, *WO* required a little more tours than ACS; this was expected and is contributed to the solitary aspect of the worm. In contrast, as the ants are only social, they converge faster to a solution. However, converging faster to a solution does not necessarily mean that this solution is the best. From Table 11.2 we observe that *WO* reached better solutions than ACS in all problems, while requiring a small number of extra tours, expressly a max deviation in tours of $\sim$3 % (from Table 11.3).

11.5 Conclusions and Future Research

In this chapter, we have introduced a novel optimization algorithm, *WO*, which is inspired by the foraging behaviors of *C. elegans* (worms). The *WO* approach and performance were illustrated using the Traveling Salesmen Problem, a known NP-hard problem. The algorithm was compared to ACS, PSO, ACOMAC, and genetic algorithm (GA). The computational tests indicated that *WO* outperformed the algorithms in all problems, as well as attained the optimal solution in all cases. Furthermore, *WO*'s computational effort was better than the GA's and similar to ACS's in terms of number of evaluated tours.

The purpose of this chapter was to introduce this novel naturally inspired algorithm and to highlight its value on a well-known problem. In future studies, *WO* will be applied to more complex and contemporary problems.

References

Arnaout J-P (2014) Worm optimization: a novel optimization algorithm inspired by C. Elegans. In: Proceedings of the 2014 international conference on industrial engineering and operations management, Indonesia, pp 2499–2505

Avery L, You YJ (2012) *C. elegans* feeding. In: WormBook D (ed) The *C. elegans* research community, WormBook. http://www.wormbook.org/chapters/www-feeding/feeding.pdf. Cited 21 May 2012

Basu S (2012) Tabu search implementation on traveling salesman problem and its variations: a literature survey. Am J Oper Res 2:163

Bersini H, Oury C, Dorigo M (1995) Hybridization of genetic algorithms. Tech. Rep. No. IRIDIA 95-22, IRIDIA, Universit Libre de Bruxelles, Belgium

Bongard J (2009) Biologically inspired computing. IEEE Comput 42:95–98

Brenner S (1974) The genetics of *Caenorhabditis elegans*. Genetics 77:71–94

Dorigo M, Birattari M (2007) Swarm intelligence. Scholarpedia 2:1462

Dorigo M, Gambardella LM (1997) Ant colonies for the travelling salesman problem. BioSystems 43:73–81

Eilon S, Watson-Gandy CDT, Christofides N (1969) Distribution management: mathematical modeling and practical analysis. Oper Res Q 20:37–53
Ferris H (2013) *Caenorhabditis elegans*. University of California, Davis. http://plpnemweb.ucdavis.edu/nemaplex/Taxadata/G900S2.htm. Cited 30 Nov 2013
Fisher RA (1960) The design of experiments. Hafner Publishing Company, New York
Hu PJ (2007) Dauer. In: WormBook (ed) The *C. elegans* research community, WormBook. http://www.wormbook.org/chapters/www-dauer/dauer.pdf. Cited 8 Aug 2007
Jabr F (2012) The connectome debate: is mapping the mind of a worm worth it? http://www.scientificamerican.com/article/c-elegans-connectome/
Krolak P, Felts W, Nelson J (1972) A man-machine approach toward solving the generalized truck-dispatching problem. Transp Sci 6:149–170
Larranaga P, Kuijpers CMH, Murga RH, Inza I, Dizdarevic S (1999) Genetic algorithms for the traveling salesman problem: a review of representations and operators. Artif Intell Rev 13:129–170
Lin F-T, Kao C-Y, Hsu C-C (1993) Applying the genetic approach to simulated annealing in solving some NP-hard problems. IEEE Trans Syst Man Cyber 23:1752–1767
Lockery S (2009) A social hub for worms. Nature 458:1124–1125
Macosko E, Pokala N, Feinberg E, Chalasani S, Butcher R, Clardy J, Bargmann C (2009) A hub-and-spoke circuit drives pheromone attraction and social behavior in *C. elegans*. Nature 458:1171–1176
Maupas E (1900) Modes et formes de reproduction des nematodes. Archives de Zoologie Exprimentale et Gnrale 8:463–624
NIST/SEMATECH (2008) e-Handbook of statistical methods. http://www.itl.nist.gov/div898/handbook/. Cited 25 Jun 2008
Oliver I, Smith D, Holland JR (1987) A study of permutation crossover operators on the travelling salesman problem. In: Grefenstette JJ (ed) Proceedings of 2nd international conference on genetic algorithms. Lawrence Erlbaum, Hillsdale, pp 224–230
Reinelt G (1994) The traveling salesman: computational solutions for TSP applications. Springer, Berlin
Reinhelt G: TSPLIB: a library of sample instances for the TSP (and related problems) from various sources and of various types, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
Ross P (1996) Taguchi techniques for quality engineering. McGraw Hill, New York
SchedulingResearch (2015) http://www.schedulingresearch.com. Accessed 1 Aug 2015
Shi X, Liang Y, Lee H, Lu C, Wang Q (2007) Particle swarm optimization-based algorithms for TSP and generalized TSP. Inform Process Lett 5:169–176
Shtonda BB, Avery L (2006) Dietary choice behavior in *Caenorhabditis elegans*. J Exp Biol 209:89–102
Taguchi G (1993) Taguchi methods: design of experiments. American Supplier Institute, Livonia, MI
Tsai CF, Tsai CW, Tseng C (2004) A new hybrid heuristic approach for solving large traveling salesman problem. Inform Sci 166:67–81
Whitley D, Starkweather T, Fuquay D (1989) Scheduling problems and travelling salesman: the genetic edge recombination operator. In: Schaffer JD (ed) Proceedings of the third international conference on genetic algorithms. Morgan Kaufmann, San Mateo, pp 133–140
Wood WB (1988) The Nematode *Caenorhabditis elegans*. Cold Spring Harbor Laboratory Press, New York
Xu J-X, Deng X (2012) Complex chemotaxis behaviors of *C. elegans* with speed regulation achieved by dynamic neural networks. In: Proceedings of the IEEE world congress on computational intelligence, Brisbane
Yang J, Wu C, Lee HP, Liang Y (2008) Solving traveling salesman problems with generalized chromosome genetic algorithm. Prog Nat Sci 18:887–892
Zhang Y, Lu H, Bargmann C (2005) Pathogenic bacteria induce aversive olfactory learning in *Caenorhabditis elegans*. Nature 438:179–184

Chapter 12
Heuristics and Simulated Annealing Algorithm for the Surgical Scheduling Problem

Gulsah Hancerliogullari, Emrah Koksalmis, and Kadir Oymen Hancerliogullari

Abstract Planning and scheduling play a very important role in health care. Effective scheduling optimizes the utilization of scarce resources such as operating rooms (ORs), devices in hospitals, and surgeons. Therefore, operations research/operations management techniques have been frequently used in health care systems management. In this chapter, we examine the surgical scheduling problem over multiple operating rooms. In order to find an optimal solution to surgical scheduling problem, mixed-integer programming (MIP) formulation of the surgical scheduling problem is presented. The model includes constraints for several operational rules and requirements found in most hospitals, and specifically minimizes the total weighted start time as a performance measure (or objective function). Since the problem is known to be an NP-hard in most of its forms, heuristic algorithms (i.e., greedy heuristics and a metaheuristic) are also introduced to find near-optimal solutions efficiently.

Keywords Health care services • Surgical scheduling • Simulated annealing • Greedy heuristic • Metaheuristic • Mathematical programming

G. Hancerliogullari (✉)
Department of Industrial Engineering, Faculty of Engineering and Natural Sciences, Istanbul Bilgi University, 34060 Istanbul, Turkey
e-mail: gulsah.hancerli@bilgi.edu.tr

E. Koksalmis
Department of Industrial Engineering, Faculty of Management, Istanbul Technical University, 34367 Istanbul, Turkey
e-mail: koksalmis@itu.edu.tr

K.O. Hancerliogullari
Department of Pediatric Surgery, Faculty of Medicine, Giresun University, 28200 Giresun, Turkey
e-mail: droymen@hotmail.com

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_12

12.1 Introduction

Health care systems are expected to provide high quality services with limited resources. Significant percentage of the gross domestic product (GDP) in both developed and developing countries belongs to health care expenses. The World Health Organization (WHO) stated that total health care spending in the USA was about 17 % of its GDP in 2011. Moreover, the Health and Human Services Department anticipates that the health share will continue its increasing trend, reaching almost 20 % of GDP by 2017 (Keehan et al. 2008). Similarly, it was stated that over 8 % of the GDP is due to the health care expenses in the United Kingdom in 2007 (Haynes 2010). According to the Health Care Financial Management Association, operating rooms comprise over 40 % of the hospital's revenue and a similar portion of its cost (HFMA 2005). In order to satisfy the growing demand for health care services, efficient use of operating rooms is crucial. Therefore, the efficiency of health care delivery system is highly dependent on efficient management of operating room. On the other hand, ineffective scheduling of operating room time usually leads to costs to hospitals and patients mainly due to delays or cancellations of surgery. Aiming to increase patient satisfaction, health care centers also try to reduce their expenditures and improve their financial assets. The key objectives of the hospital authorities are to utilize resources optimally and to plan the surgery at the right time, at the right operating room.

The construction of a schedule that satisfies all operational rules and needs in hospitals, while simultaneously fulfilling as many of the wishes and requirements of the staff and patients is an important but extremely difficult task. In most hospitals this duty is left to each surgical department and the current practice is to replicate the schedules of previous weeks with minor changes to accommodate newly developed conditions. In recent years, changes occur more frequently and patching of what had been developed previously is not always the best strategy. Under these conditions, and in the light of development done both in hardware and software technologies, the scientific community continues to work on the problem so as to develop formal and automated procedures for constructing efficient and desirable schedules.

The surgical scheduling problem is defined as the process of assigning surgical operations to specific time periods (mostly throughout the 5 working days of the week) and specific ORs suitable for the patients with respect to the needs of each surgery. For every health care center, the objective is always the construction of effective and satisfactory weekly schedules. A schedule is considered to be effective when it is feasible and realized by the center, while it is considered to be satisfactory when it carries certain quality characteristics that keep its users satisfied at least to a certain degree.

12.2 Approaches in Literature

Operations Research/Operations Management techniques have been commonly used in health care systems management, and a large body of literature has reported on the operating theater (OT) planning and scheduling. The surgical scheduling problem is known to be an NP-hard problem in most of its forms, meaning that if all combinations were to be examined, the solution time would increase considerably with slight increase in problem size. Similar to many other problems in the area of combinatorial optimization, the surgical scheduling problem has been approached by numerous well-known methods of operational research and computer science.

In the literature, deterministic and stochastic mathematical programming models, queuing models, simulation models, and heuristic approaches have all been widely addressed in order to investigate operating room scheduling. In this chapter, we review studies that pertain to deterministic programming approach for surgical operation scheduling problem. However, a more comprehensive review can be found in Cardoen et al. (2010) and Guerriero and Guido (2011), in which they provide broad overview and classification of the OT planning and scheduling. A detailed taxonomy has been proposed based on several areas related to the problem setting (e.g., performance measures or patient classes) and the technical features (e.g., solution technique and uncertainty incorporation). They stated that most of the research that considers deterministic methodologies focus on planning and scheduling of elective cases at an operational level of control. Moreover, they observed that only few researches address the implementation in practice.

Mathematical programming models (Ogulata and Erol 2003; Blake et al. 2002; Blake and Carter 2002; Ozkarahan 2000) and simulation (Dexter 2000; Schmitz and Kwak 1972; Kuzdrall et al. 1974; Vasilakis et al. 2007) are the most common approaches applied to the surgical operation planning problems. Mathematical programming (especially, integer and mixed-integer programming) models have shown to be useful in capacity planning or resource allocation in complex health care systems. Cardoen et al. (2009) stated a multi-objective surgical case sequencing problem in which the order of patients in the operating rooms of a freestanding ambulatory unit had to be determined. Roland et al. (2006) proposed a genetic algorithm (GA) so as to minimize the costs related to operating room openings and overtime. Specifically, the stated problem, which is similar to the well-known resource-constrained project scheduling problem, determines the date and starting time of a set of surgeries in the OR. The performance of the proposed GA is validated through a comparison with a mixed-integer programming (MIP) approach.

Although satisfying the growing demand for health care services and efficient use of ORs is highly crucial, research on optimum scheduling of hospital operations is limited. For example, a goal programming model has been proposed for the assignment of surgical operations among multiple operating rooms (Ozkarahan 2000). The hierarchical multiple criteria mathematical programming is proposed in

order to assign surgeon groups and dates to surgical operations optimally (Ogulata and Erol 2003). An MIP model is introduced in Vissers et al. (2005) to generate a master operating theater schedule for thoracic surgery. Fei et al. (2010a) proposed a branch-and-price algorithm in order to assign a set of surgeries to operating rooms. Their objective was to minimize the total operating cost. Fei et al. (2010b) used the open scheduling strategy to construct a weekly surgery schedule. The problem was solved in two steps, firstly surgery date for each patient, then the sequence of surgeries in each OR is determined. Roland et al. (2010) proposed an approximate algorithm, which is based on GA in order to solve the resource constraint project scheduling problem (RCPSP). Hanset et al. (2010) developed a constraint programming approach to solve a daily scheduling problem in which they consider human and material aspects, and they compared the results with the MIP model developed for the same problem (Dexter et al. 1999).

Regarding the objective function, most of the studies focus on maximizing OR utilization (Ozkarahan 2000; Dexter et al. 1999; Dexter and Traub 2002). On the other hand, Dexter et al. (2002) considered not only OR but also the hospital beds in the objective function. Following this idea, Beliën and Demeulemeester (2007) developed heuristics to construct master surgical scheduling (MSS) in which the objective is minimizing the expected total bed shortage.

Continuous growth in health care operation volumes worldwide and increasing delay costs continue to motivate research related to health care operations management. Surgical scheduling problems, in particular, have motivated a numerous deal of academic research since ORs constitute a major bottleneck at hospitals.

In this study, we provide an illustrative example in order to demonstrate how the operations research techniques can be used in the surgical scheduling problem. We provide an MIP formulation and approximate algorithms to solve the surgical scheduling problem in case of multiple ORs and multiple surgical operations while taking into account real-world constraints. Specifically, we make the following contributions: First, contrary to most existing studies that ignore the priorities of the surgical operations, we model the surgical scheduling problem while considering different cases and categories of patients based on their surgical cases (i.e., elective, urgent) and ages (i.e., baby, child, adult). In general, an urgent case has more risk than an elective case; therefore, for instance, we assign higher priority to the urgent surgical operations. Second, even though the ORs are assumed to be identical, we consider that each surgical operation may require specific equipment. Whenever an operation is completed, some setup work has to be done before the next surgical operation starts, such as changing equipment. Therefore, we consider the sequence-dependent setup times between surgical operations which increases the complexity of the problem. Third, we provide problem specific greedy heuristics (i.e., shortest processing time, weighted shortest processing time, etc.), and integrated them into Simulated Annealing (SA) algorithm to improve initially constructed solutions.

12.3 Problem Statement

ORs are considered a scarce resource in the surgical scheduling problem, and their availability is defined by the hospital. Some ORs may be considered available during all periods, while some others have limited availability. In some hospitals, one department may operate in an OR some days of the week and other departments the rest of the periods. Therefore, it is common for several departments to share the same OR.

Construction of a daily/weekly schedule for a health care center is a monotonous activity on which administrators typically spend several man-hours. In this effort, schedulers follow several rules, some of which are so important that they may never be violated. For instance, in a timetable, collisions are not acceptable. A collision occurs when at least two surgical operations are scheduled at the same period in the same OR, or when at least two ORs are allocated to the same surgical operation and to the same patient. A schedule has to be complete, which happens when all surgeries planned for every patient appear in the timetable, with the right amount of time periods for every surgery and every portion of each surgery.

In this chapter, we study the problem of scheduling a set of surgical operations (i.e., elective and urgent surgeries) to multiple ORs in a health care center. The scheduling decision includes the assignment of surgical operations to ORs, and the operation sequence in each OR. Various types of surgical operations may necessitate various resources and equipment. In general, most facility resources such as ORs, specified equipments are multi-functional and can accommodate different types of surgical cases. While some resources and equipment can be shared among several ORs, some of them are dedicated to particular ORs. Moreover, each surgical operation is constrained by the resources and equipment associated with the OR.

In the surgical scheduling problem, we consider a set J of surgical operations ($j = 1, \ldots, n$) to be performed in an OR. Each surgical operation j should be performed after a given earliest start time, r_j and before a given latest start time, d_j, in which their values vary by several factors such as the availability and the condition of the patient, the availability of the resource that is going to be used (e.g., the schedule of the surgeon who is going to operate the surgical operation), the time-window set by the health care center authorities. A priority weight, w_j, is assigned to each surgical operation and its value depends on the case of surgical operation (elective, urgent) and the category of age of patients (baby, child, adult). For instance, urgent cases always have higher priority over elective cases regardless of the age of patients; or babies have higher priority weight over adults when they belong to same case (elective, urgent) since the hunger tolerance of the babies is lower. Different types of surgical operations contribute to different surgery times. The duration of each surgical operation, p_j, is dependent on the type of the procedure, and can last from minutes to hours. Whenever an operation is completed, some setup work has to be done before the next surgical operation starts; for instance, cleaning up the OR, changing equipment, refilling sterilization resources, and getting proper surgeons, nurses, and anesthesiologists. A sequence-dependent setup time, $s_{j_1 j_2}$, is enforced

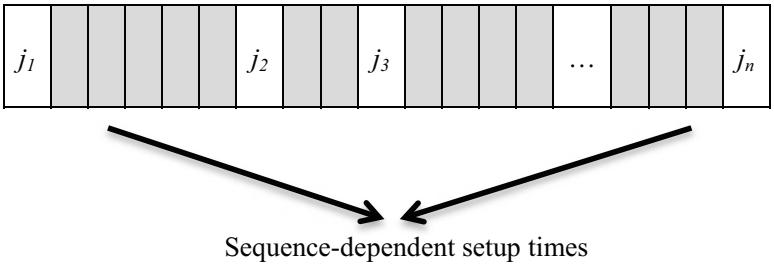


Fig. 12.1 The schedule of a set of surgical operations with sequence-dependent setup time

if two successive surgical operations j_1 and j_2 are assigned to the same OR, as shown in Fig. 12.1. That is, the setup time required after j_1 is dependent on the next surgical operation. For instance, in Fig. 12.1, the setup time between surgical operation j_1 and j_2 is 5 units; however, if another surgical operation was scheduled right after j_1, the setup time would differ. Moreover, scheduling procedure j_2 after j_1 may require a different setup time if j_1 was scheduled after j_2. There is a set M of ORs ($i = 1, \ldots, m$) available each day. Each OR i has a maximum available time to be used, a_i. The start time of the surgical operation j is t_j, and the scheduling goal is to minimize the total weighted start times, which in practice tends to start the procedures on time.

Finally, the following realistic assumptions are made as follows:

1. Only one OR can be allocated to a surgical operation at a time.
2. A surgical operation cannot be interrupted or stopped.
3. The duration of a surgical operation has been predetermined prior to the scheduling.
4. The priorities of surgical operations are not the same.
5. Before a surgical operation starts, some setup work needs to be done, such as transfer of resources, preparing required equipment, sterilization, etc.
6. The setup times are sequence-dependent, and deterministic.

12.4 Mathematical Programming Formulation

We present an MIP formulation to construct surgeries schedule. This MIP has been adapted from the literature for a different application (see, for example, Al-Salem et al. 2012; Hancerliogullari et al. (2013)). The equations and inequalities of the MIP model presented here may differ from a hospital to another to reflect the special requirements imposed by each one of them. Given a set of n surgical operations and m ORs, it is required to simultaneously assign the surgical operations to ORs and generate the optimum schedule for each OR with respect to an objective function as determined by the health care authority (minimizing the total weighted start times in this case).

12.4.1 Notation

Sets

J: Set of surgical operations
M: Set of ORs

Parameters

p_j: the duration of surgical operation j, $\forall j \in J$
r_j: the earliest start time of surgical operation j, $\forall j \in J$
d_j: the latest start time of surgical operation j, $\forall j \in J$
a_i: the length of time that OR i is planned to be available, $\forall i \in M$
e_j: the case of surgical operation j, being an elective surgical case or an urgent case, $\forall j \in J$
c_j: category of surgical operation j, based on patient's age, e.g. baby, child, or adult, $\forall j \in J$
w_j: the priority weight assigned to surgical operation j based on its case and its category,$\forall \in J$. In particular higher priority is assigned to urgent cases over elective cases and to babies over child and adults due to hunger tolerance.
$s_{j_1 j_2}$: the setup time required between surgical operation j_1 and j_2 if they are assigned to same OR and are, respectively, the leading and the following surgical operation, $\forall j_1, j_2 \in J,\ j_1 \neq j_2$

Decision Variables

$x_{ij} = 1$ if surgical operation j is assigned to OR i, $\forall i \in M,\ \forall j \in J$ and $= 0$ otherwise
$z_{j_1 j_2} = 1$ if surgical operation j_1 and j_2 are assigned to the same operating room, $\forall j_1, j_2 \in J,\ j_1 \neq j_2$, and $= 0$ otherwise
t_j: the start time of surgical operation j, $\forall j \in J$

12.4.2 Model Formulation

$$\text{Minimize} \sum\nolimits_{j \in J} w_j t_j \tag{12.1}$$

$$\sum\nolimits_{i \in M} x_{ij} = 1,\ \forall j \in J \tag{12.2}$$

$$\sum\nolimits_{j \in J} p_j * x_{ij} \leq a_i,\ \forall i \in M \tag{12.3}$$

$$t_{j_2} \geq t_{j_1} + p_{j_1} + s_{j_1 j_2} - \left(1 - z_{j_1 j_2}\right)\left(d_{j_1} + p_{j_1} + s_{j_1 j_2}\right),\ \forall j_1, j_2 \in J,\ j_1 \neq j_2 \tag{12.4}$$

$$z_{j_1 j_2} + z_{j_2 j_1} \geq x_{ij_1} + x_{ij_2} - 1, \ \forall i \in M, \ \forall j_1, j_2 \in J, \ j_1 < j_2 \tag{12.5}$$

$$r_j \leq t_j \leq d_j, \ \ \forall j \in J \tag{12.6}$$

$$x_{ij} \in \{0, \ 1\} \tag{12.7}$$

$$z_{j_1 j_2} \in \{0, \ 1\} \tag{12.8}$$

$$t_j \geq 0, \ \forall j \in J \tag{12.9}$$

The objective function (12.1) is to minimize the total weighted start times. Constraint (12.2) indicates that every surgical operation is assigned to an OR. Constraint (12.3) guarantees that total duration of surgical operations in an OR cannot exceed the available operating time of the room. Constraint (12.4) calculates the value of the start time decision variable while ensuring a procedure's duration and proper sequence-dependent setup time between any pair of surgical operations if they are assigned to the same operating room. For instance, when two surgical operations are assigned to the same operating room ($z_{j_1 j_2}$ is 1), the start time of the following surgical operation must consider not only the start time and the duration of the leading surgical operation but also the sequence-dependent setup time. On the other hand, when $z_{j_1 j_2}$ is 0, which means that surgical operation j_1 and j_2 are not assigned to the same operating room, t_{j_2} of the constraint is enforced to be greater than zero or a negative value since $t_{j_1} \leq d_{j_1}$. Constraint (12.5) initiates the sequencing variables between any pair of surgical operation that are assigned to the same OR. For example, if surgical operation j_1 and j_2 are assigned to the same operating room, i.e., x_{ij_1} and x_{ij_2} are equal to 1, either $z_{j_1 j_2}$ or $z_{j_2 j_1}$ must be 1. Constraint (12.6) determines the time-window limitations. Constraints (12.7)–(12.8) define x_{ij} and $z_{j_1 j_2}$ to be binary decision variables. Constraint (12.9) states that the start time of all surgical operations should be nonnegative.

12.5 Approximate Algorithms

Minimizing the total weighted start time with multiple ORs is NP-hard, which leads us to provide appropriate solution methods, such as greedy heuristics and metaheuristics to obtain good quality solutions in reasonable computational times. We study the performance of several greedy heuristics listed below:

Table 12.1 Sample priority weights, durations, earliest start time, latest start time, and the sequence-dependent setup times

SO	w_j	p_j (min)	r_j (min)	d_j (min)	SDST (min)					
						1	2	3	4	5
1	3	40	10	300	1	–	40	50	60	50
2	2	60	10	300	2	45	–	60	70	60
3	3	100	10	300	3	55	60	–	50	60
4	1	40	10	300	4	55	70	50	–	60
5	4	80	10	300	5	65	55	45	55	–

SO surgical operation, p_j duration, w_j priority weight, r_j earliest start time, d_j latest start time, *SDST* sequence-dependent setup time

Fig. 12.2 Sample surgery schedule using SPT rule

12.5.1 Smallest Processing Time

Whenever an OR is free and the surgical operation is ready to be scheduled, the smallest processing time (SPT) rule assigns first the surgical operation with the SPT among all unassigned surgical operations. Ties can be broken arbitrarily. The computation time needed to order the surgical operations according to SPT rule is $O(n(\log(n))$ time. The rule is explained through the example stated below:

A sample priority weights, surgical operation durations, earliest start time, latest start time, and the sequence-dependent setup times of five surgical operations are provided in Table 12.1. According to the SPT rule, a surgical operation to be assigned first is the one that has the shortest duration. The remaining surgical operations are scheduled according to the ascending surgical durations to the ORs on which their operation can start earlier. The constructed schedule for a given example is provided in Fig. 12.2.

12.5.2 Higher Priority Earlier

By sequencing surgical operations in decreasing order of the priority weights, w_j, this greedy heuristic allows us to schedule first the surgical operation that has urgency to the first available OR if available and the hard constraints are satisfied (e.g., a surgical operation should start within the earliest start time–latest start time

Fig. 12.3 Sample surgery schedule using HPE rule

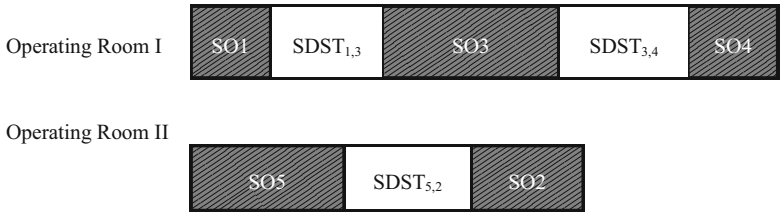


Fig. 12.4 Sample surgery schedule using WSPT rule

window). The computational complexity of the higher priority earlier (HPE) rule is $O(n(\log(n))$.

According to HPE rule, the surgical operation to be assigned first is the one that has the highest priority weight. The remaining surgical operations are scheduled according to the descending surgical priority weights to the ORs on which their operation can start earlier. The constructed schedule for the example presented earlier is provided in Fig. 12.3.

12.5.3 Weighted Shortest Processing Time First (Ratio Rule)

The priority weight w_j of surgical operation j may be regarded as an importance factor; it may represent the value already added to surgical operation j (higher value of w_j is higher priority). The surgical scheduling problem gives rise to one of the better known greedy rules in scheduling theory, the so-called weighted shortest processing time (duration) first (WSPT) rule. According to this greedy rule, the surgical operations are ordered in decreasing order of w_j/p_j, the surgical operation to be assigned first is the one that has the highest ratio (w_j/p_j) value. The remaining surgical operations are scheduled according to the descending values to the ORs on which their operation can start earlier. In the example above, the order of surgical operations will be 1, 5, 3, 2, 4, and the constructed schedule will be as shown in Fig. 12.4. The computational complexity of the WSPT rule, which is dependent on the sorting procedure, is $O(n(\log(n))$.

12.5.4 Simulated Annealing (SA)

SA algorithm is one of the robust metaheuristics designed to solve combinatorial optimization problems such as the surgical scheduling problem. The algorithm mimics the annealing process used in metallurgy which is based on the work of Metropolis et al. (1956). Starting with an initial solution, one replaces the current solution (θ) by a neighbor solution (θ') with a probability at each iteration. Providing a means of escaping from local optimum, the metaheuristic sometimes accepts a worse neighborhood move with an acceptance probability P_a that is defined as $e^{-\Delta/T}$, where Δ is the difference in the objective function values of the current solution and candidate solution, and T is a temperature control parameter. As is the case in the annealing process, the temperature (T) is decreased gradually throughout the process. The performance of the algorithm depends on neighborhood search and several parameters, namely the maximum number of iterations ($t_{\max}$), maximum number of inner loop iterations ($i_{\max}$), temperature cooling coefficient (α), and initial temperature coefficient (k). In order to have more flexible initial temperature values, as in Hancerliogullari et al. (2013), the initial temperature is defined as a function of the objective function value of the current solution, which can be the initial solution obtained by the greedy heuristics. The values of the parameters can be fine-tuned using experimental design methodology. While the SA goes through k drops in temperature, at each temperature, it explores the neighborhood of the current solution. Swap 1, Swap 2, and Swap 3 functions are used to perturb the current solution locally and are selected randomly with equal probability. Swap 1 randomly selects two surgical operations j_1 and j_2 from the set of operations that are assigned to the same OR and have the same priority weight (i.e., $z_{j_1 j_2} = 1$ or $z_{j_2 j_1} = 1$, and $w_{j_1} = w_{j_2}$). Swap 2 swaps two surgical operations j_1 and j_2 that are randomly from the set of operations that are assigned to the same OR regardless of priority (i.e., $z_{j_1 j_2} = 1$ or $z_{j_2 j_1} = 1$). Finally, Swap 3 exchanges a randomly selected surgical operation j_1 with another randomly selected surgical operation j_2, and it is applied to all surgical operations across the ORs. The pseudo code describing the metaheuristic is presented below.

S: search area
θ^*: best solution
$f(\theta)$: objective function value of the current solution
$N(\theta)$: neighborhood of θ
M: memory set of current best solution and objective function value
i: inner loop iteration counter
c: iteration counter

begin Greedy initialization (generate an initial solution θ from S)
Calculate $f(\theta)$, initialize *Memory* $M = \{(\theta, f(\theta))\}$
Set iteration counters $i = 0$, $c = 0$
Set initial temperature $T = k.f(\theta)$
while $c < t_{max}$

while $i < i_{max}$
Choose $\theta' \epsilon N(\theta) \subseteq S$ *where* $M = \{(\theta, f(\theta'))\}$, do neighborhood search
if $z_{j_1 j_2} = 1$ *or* $z_{j_2 j_1} = 1$, and $w_{j_1} = w_{j_2}$, use Swap 1
else if $z_{j_1 j_2} = 1$ *or* $z_{j_2 j_1} = 1$, use Swap 2
else use Swap 3
Calculate $f(\theta')$
if $f(\theta') - f(\theta) \leq 0$ or $rand\ [0,1] \leq e^{-\frac{\Delta\theta}{T}}$, *where* $\Delta\theta = f(\theta') - f(\theta)$ **then**
accept the new neighbor θ' and set $M = \{(\theta, f(\theta'))\}$
end if
$i = i+1$
$c = c+1$
end while
Update temperature $T = \alpha.T$
end while
Calculate $f(\theta^*)$
end

12.6 Computational Experiments

In this section, we evaluate the computational performance of the MIP model, the greedy heuristics and the metaheuristic on randomly generated instances. We generate 90 instances with different sizes as follows: the number of surgical operations $J = 15, 30, 50$ and the number of ORs $M = 4, 5, 6$. In order to test the performances of the solution methodologies, the data are generated as follows: The surgical operation cases are randomly generated with equal probability of 1/2 to represent urgent and elective surgeries. The (age) categories of surgical operations/patients are randomly generated with equal probability of 1/3 to represent adults, children, or babies. The priority weights for surgical operations are dependent on the case of surgical operation (i.e., elective or urgent) and the categories of surgical operation/patient (i.e., baby, child, or adult). Furthermore, urgent cases always have higher priority weights over elective cases. The priority weight for an elective surgical operation is 1, 2, and 3 for an adult, a child, and a baby, respectively; and it is 4, 5, and 6 for an urgent surgical operation for an adult, a child, and a baby, respectively. According to Jebali et al. (2006), lognormal distributions are suitable to generate the operating times (i.e., duration). Therefore, we generate the surgery durations by following lognormal distributions with a mean of 120 min and a standard deviation of 60 min. The earliest and latest start times of the surgical operations and the length of the time that OR i is planned to be available are randomly generated using a discrete uniform distribution. The sequence-dependent setup times for surgical operations range between 15 and 35 min depending on the type of surgical operation.

Table 12.2 The results for MIP model

Instance set	# of surgical operations	# of operating rooms	MIP				
			CPU (min)	Best solution so far	# of constraints	# of variables	# of iterations
1	15	4	>30	1844	675	286	129,367,260
2	15	5	>30	1279	780	301	72,922,793
3	15	6	>30	898	885	316	66,769,882
4	30	4	>30	12,090	2700	1021	45,407,445
5	30	5	>30	8678	3135	2051	44,223,441
6	30	6	>30	7463	3570	1081	44,051,307
7	50	4	>30	33,260	7500	2701	28,489,668
8	50	5	>30	25,550	8725	2751	26,692,546
9	50	6	>30	21,355	9950	2801	24,079,034

The MIP model is implemented and solved using IBM ILOG CPLEX CP Optimizer 12.5, and the approximate algorithms are coded in C using Visual Studio and run on Windows 7 with Intel Core i5-3330S processor, with 2.70 GHz CPU and 8 GB of memory.

Table 12.2 shows the performance of the MIP model and it lists the number of surgical operations, number of ORs, CPU time, average of best solution (objective function value) found so far, number of constraints and the number of variables. As the number of surgical operations increases with the same number of ORs, the objective function value, the number of constraints, the number of variables, and the number of iterations, which is the number of times executed so far during the current optimization to solve the node relaxations, increase. In contrast, as the number of ORs increases for the same number of surgical operations, the number of constraints and the number of variables increase, while the objective function value and the number of iterations decrease. The MIP model failed to solve our instances using CPLEX within a time limit of 30 CPU minutes. Therefore, approximate algorithms are needed to solve the various problem instances in acceptable amount of time.

The performance of the greedy heuristics, SPT, HPE, WSPT, and the metaheuristic SA provided in Tables 12.3 and 12.4, is measured in terms of average percentage optimality gap and average CPU time by averaging the values for 10 instances for each instance set (surgical operations-OR combination). The optimality gap is calculated via Eq. (12.10) for each test problem.

$$Optimality\ Gap = \frac{Objective\ function\ value_{Alg} - Objective\ function\ value_{Optimal}}{Objective\ function\ value_{Optimal}} \tag{12.10}$$

where Objective function $\text{value}_{\text{Alg}}$ is the total weighted start time of the provided algorithms and Objective function $\text{value}_{\text{Optimal}}$ is the optimal solution's total weighted start time for each test problem. For the cases, where optimal solutions

Table 12.3 The results for the greedy algorithms

Instance	# of surgical operations	# of operating rooms	SPT		HPE		WSPT	
			Gap	CPU (second)	Gap	CPU (second)	Gap	CPU (second)
1	15	4	0.52	0.00	0.49	0.00	0.23	0.00
2	15	5	2.03	0.00	0.71	0.00	1.73	0.00
3	15	6	1.46	0.00	1.50	0.00	1.46	0.00
4	30	4	0.59	0.00	0.84	0.00	0.19	0.00
5	30	5	1.07	0.00	1.23	0.00	0.86	0.00
6	30	6	1.43	0.00	1.49	0.00	0.87	0.00
7	50	4	0.68	0.00	0.55	0.00	0.31	0.00
8	50	5	0.92	0.00	0.79	0.00	0.39	0.00
9	50	6	0.89	0.00	0.82	0.00	0.49	0.00

Table 12.4 The results for the simulated annealing algorithm

Instance	# of surgical operations	# of operating rooms	SA_{SPT}		SA_{HPE}		SA_{WSPT}	
			Gap	CPU (second)	Gap	CPU (second)	Gap	CPU (second)
1	15	4	0.06	0.64	0.06	0.67	0.06	0.64
2	15	5	0.24	0.77	0.24	0.75	0.24	0.74
3	15	6	0.81	0.86	0.81	0.88	0.81	0.85
4	30	4	0.05	0.73	0.05	0.75	0.05	0.72
5	30	5	0.23	0.82	0.22	0.82	0.23	0.82
6	30	6	0.50	0.93	0.51	0.94	0.50	0.91
7	50	4	0.06	0.95	0.06	0.95	0.06	0.95
8	50	5	0.21	0.96	0.21	0.98	0.23	0.96
9	50	6	0.06	1.94	0.05	1.96	0.06	1.96

could not be found in predefined time limit, we keep the record of "best solution found so far," and use this value instead of the optimal solution.

The greedy algorithms yield feasible solutions to all instances in our testbed along with substantial computational savings over solving the MIP model using CPLEX. For example, these three greedy heuristics solved instances with $(n, m) = (15, 4)$ in very short time (0.00 s), and exhibited an optimality gap that ranged from 0.23 to 0.52. It is notable that whereas the solution of the MIP model with CPLEX reached the computational time limit of 30 CPU minutes for most of these instances, the greedy heuristics solved the instances in very short time (0.00 s) on average. This demonstrates the usefulness of using the greedy algorithms in surgical scheduling problem. The WSPT rule, which takes into account both the priority weight and the duration of the surgical operation in assigning operations to ORs, performs better than SPT and HPE rules in terms of average optimality gap and takes similar CPU.

The results of SA with the greedy heuristics used in the initialization phase are provided in Table 12.4. When the SA algorithm uses the solution obtained from the SPT greedy rule as an initial solution, the integrated algorithm is called SA_{SPT}. Similar convention is used when the other two greedy heuristics provide the initial solutions for the SA algorithms. The performance of the SA with different initial solutions is evaluated in terms of average optimality gap and average CPU for all problem instances. The results show that the SA provides significant computational savings compared to the MIP model, and it generates feasible solution to all instances. The average optimality gap and average CPU of the SA_{SPT}, SA_{HPE}, and SA_{WSPT} are quite similar which indicates that for the given problem instances, the SA performance is not affected by the initial solutions generated by greedy rules.

The general trend of the results indicates that the SA algorithm which uses greedy heuristics as initial solutions performs better than the greedy algorithms by themselves (e.g., SA_{SPT} is superior to SPT), in terms of average optimality gap which is expected without increasing the CPU significantly. For example, SA_{SPT} solves the instances with $(n, m)=(15, 4)$ in less than a second, and the optimality gap is reduced to 0.06 %. All instances and solutions are available at www.SchedulingResearch.com.

12.7 Conclusion

ORs are regarded as the hospital's engine because they are considered among the most costly hospital facilities. They comprise almost half of the hospital's revenue and a similar portion of its cost (HFMA 2005). Therefore, efficient use of OR is very important in order to satisfy the growing demand for health care services. Ineffective and incorrect scheduling of surgical operations usually results in costs to hospitals and patients mainly because of delays or cancellations of surgery. The objective of health care centers is both increasing the patient satisfaction and improving their financial assets by decreasing their expenditures due to inefficiencies. Consequently, efficiency of health care delivery system is highly dependent on efficient planning and scheduling of surgical operations.

In this chapter, we demonstrated how operations research techniques can be used in health care systems management by specifically studying the surgical scheduling problem. We have examined the real-world application of the surgical scheduling problem in case of multiple operating rooms and multiple surgical operations, where the priorities of the surgical operations are taken into consideration. In this study, the priorities are dependent on the type of surgical cases (i.e., elective, urgent) and ages (i.e., baby, child, adult). Moreover, contrary to most existing studies, we consider that each surgical operation may require specific resource such as equipment; therefore, whenever an operation is completed, sequence-dependent setup time is enforced before the next surgical operation starts.

We started with providing an MIP formulation to model the problem and obtain optimal schedules. According to the computational study, even within a time limit of 1800 CPU seconds, it was not possible to solve the MIP and find

exact (optimal) solutions to this challenging problem. In literature, the problem is known to be an NP-hard; hence, greedy heuristics (SPT, HPE, and WSPT) and a metaheuristic (simulated annealing algorithm) are provided to find near-optimal solutions efficiently. The results indicate that even though the performance of the greedy algorithms is similar in terms of average CPU times, which are very close to zero, WSPT outperforms both SPT and HPE with low average optimality gap. The performance analysis of the simulated annealing metaheuristic indicates that it is an effective algorithm where its solution quality is better than the greedy algorithm by itself without increasing the computational time significantly. The overall results show that the value of the proposed greedy heuristics and the simulated annealing algorithm becomes more evident as the problem size increases.

References

Al-Salem A, Farhadi F, Kharbeche M, Ghoniem A (2012) Multiple-runway aircraft sequencing problems using mixed-integer programming. Proceedings of the IIE Annual Conference

Beliën J, Demeulemeester E (2007) Building cyclic master surgery schedules with leveled resulting bed occupancy. Eur J Oper Res 176(2):1185–1204

Blake JT, Carter M (2002) A goal programming approach to strategic resource allocation in acute care hospitals. Eur J Oper Res 140:541–561

Blake JT, Dexter F, Donald K (2002) Operating Room Manager's use of integer programming for assigning block time to surgical groups: a case study. Anesth Analg 94:143–148

Cardoen B, Demeulemeester E, Beliën J (2009) Optimizing a multiple objective surgical case sequencing problem. Int J Prod Econ 119:354–366

Cardoen B, Demeulemeester E, Belien J (2010) Operating room planning and scheduling: a literature review. Eur J Oper Res 201:921–932

Dexter F (2000) A strategy to decide whether to move the last case of the day in an operating room to other empty operating room to decrease overtime labor costs. Anesth Analg 91:925–928

Dexter F, Traub RD (2002) How to schedule elective surgical cases into specific operating rooms to maximize the efficiency of use of operating room time. Anesth Analg 94:933–942

Dexter F, Macario A, Traub RD (1999) Which algorithm for scheduling add-on elective cases maximizes operating room utilization? Anesthesiology 91:1491–1500

Dexter F, Blake JT, Penning DH, Sloan B, Chung P, Lubarsky DA (2002) Use of linear programming to estimate impact of changes in a hospital's operating room time allocation on perioperative variable costs. Anesthesiology 96(3):718–724

Fei H, Chu C, Meskens N, Artiba A (2010a) Solving surgical cases assignment problem by a branch-and-price approach. Int J Prod Econ 112(1):96–108

Fei H, Chu C, Meskens N, Artiba A (2010b) A planning and scheduling problem for an operating theatre using an open scheduling strategy. Comput Ind Eng 58(2):221–230

Guerriero F, Guido R (2011) Operational research in the management of the operating theatre: a survey. Health Care Manag Sci 14:89–114

Hancerliogullari G, Rabadi G, Al-Salem AH, Kharbeche M (2013) Greedy algorithms and metaheuristics for a multiple runway combined arrival-departure aircraft sequencing problem. J Air Transp Manag 32:39–48

Hanset A, Meskens N, Duvivier D (2010) Using constraint programming to schedule an operating theatre. Proceedings of the IEEE Workshop on Health Care, pp 1–6

Haynes J (2010) UK Centre for the measurement of government activity expenditure on healthcare in the UK. The Office of Public Sector Information

HFMA (2005) Achieving operating room efficiency through process integration. Technical Report. Health Care Financial Management Association

Jebali A, Haj A, Ladet P (2006) Operating rooms scheduling. Int J Prod Econ 99:52–62

Keehan S, Sisko A, Truffer C, Smith S, Cowan C, Poisal J, Clemens MK, National Health Expenditure Accounts Projections Team (2008) Health spending projections through 2015: the baby-boom generation is coming to medicare. Health Affairs Web Exclusive

Kuzdrall PJ, Kwak NK, Schmitz HH (1974) The Monte Carlo simulation of operating-room and recovery-room usage. Oper Res 22(2):434–440

Metropolis N, Rosenbluth A, Rosenbluth M, Teller A, Teller E (1956) Equation of state calculations by fast computing machines. J Chem Phys 21:1087–1092

Ogulata SN, Erol R (2003) A hierarchical multiple criteria mathematical programming approach for scheduling general surgery operations in large hospitals. J Med Syst 27(3):259–270

Ozkarahan I (2000) Allocation of surgeries to operating rooms by goal programming. J Med Syst 24(6):339–378

Pinedo M (2008) Scheduling theory, algorithms, and systems, 3rd edn. Springer, New York

Roland B, Martinelly CD, Riane F (2006) Operating theatre optimization: a resource-constrained based solving approach. Proceedings of the international conference on service systems and service management

Roland B, Martinelly CD, Riane F, Pochet Y (2010) Scheduling an operating theatre under human resource constraints. Comput Ind Eng 58(2):212–220

Schmitz H, Kwak NK (1972) Monte Carlo simulation of operating-room and recovery-room usage. Oper Res 20(6):1171–1180

Vasilakis C, Sobolev BG, Kuramoto L, Levy AR (2007) A simulation study of scheduling clinical appointments in surgical care. J Oper Res Soc 58:202–211

Vissers J, Adan I, Bekkers JA (2005) Patient mix optimization in tactical cardiothoracic surgery planning: a case study. IMA J Manag Math 16(3):281–304

WHO (2011) World health statistics 2011. World Health Organization, Geneva

Chapter 13
Product Wheels in Manufacturing Operations Planning

J. Bennett Foster and Peter L. King

Abstract This chapter discusses a production planning method known as "product wheels." We define "product wheels," discuss how they are used, and show the value the technique provides to production operations. We look at the importance of product families in planning production, particularly where set-up costs and time are critical. We examine the question of product sequencing—and why that aspect of manufacturing planning is seldom as difficult and data intensive as the mathematics (e.g., traveling salesman problem) might imply. The chapter analyzes how variants of Economic Order Quantity ("EOQ") and "EOQ with Joint Replenishment" [E.A. Silver heuristic (1976)] can be used (balancing costs of cycle stock inventory versus transitions) to get early results that lead us towards the formulation of a cost-effective wheel. We also look at the problem of balancing wheels for capacity feasibility when product campaigns cycle at different frequencies.

Keywords Product wheels • Production planning • Setup cost

13.1 Introduction

Many manufacturing operations must produce a wide variety of products or materials, which poses a number of challenging questions:

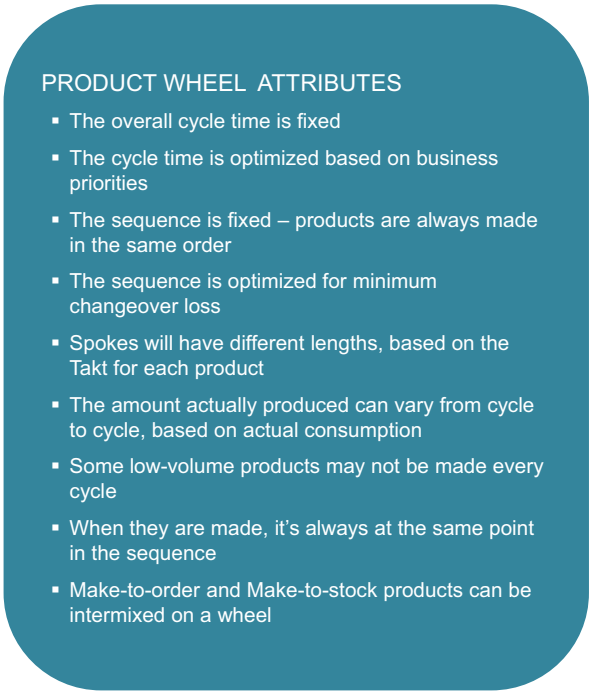
1. What should I make next, after I'm finished making the current product?
2. Is there an optimum product sequence to follow which will reduce transition cost and time?
3. If I decide to follow a fixed sequence, is there an optimum cycle over which I should produce all my "runners" (high volume products)?

J.B. Foster (✉)
DuPont Company, Wilmington, DE, USA
e-mail: jbennett.foster@gmail.com

P.L. King
Lean Dynamics, Inc, Newark, DE, USA
e-mail: PeterKing@LeanDynamics.us

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2_13

4. How frequently should I produce my "repeaters" and "strangers" (medium and low volume products)?
5. How do I best integrate the scheduling of make-to-order (MTO) products into my largely make-to-stock (MTS) schedule?



Many large companies have found that the best way to deal with all of these concerns in an integrated way is through the use of product wheel scheduling (King and King 2013).

There is also the challenge to level production. It is a very well-understood principle within the lean manufacturing community that production should be levelled, i.e., that peaks and valleys should be minimized, to minimize the waste of resources needed during production peaks during the valleys, a concept called "Heijunka" (Womack and Jones 1996). It is also a core principle that production be synchronized with customer demand, known as "Takt." This presents operations with an apparent paradox. They need to produce to customer demand, which has variation, while at the same time removing variation from the rate of production. These conflicting forces can be reconciled by integrating customer demand over some reasonably short time and levelling production to that demand. But what is a reasonable time period? Product wheels provide a very effective way to determine the optimum period.

Another reason product wheels are used in many operations is that wheels make it easy to order raw materials and plan for meeting demand. Modern ERP systems can

certainly handle material orders and sales planning without depending on regular production cycles. However both vendors and customers often become accustomed to a certain rhythm of production and the likelihood of error is reduced when that rhythm continues. "Economy of repetition" is part of the rationale for product wheels. Technically, repetition may not be necessary—but because of human nature it's often a good idea.

13.2 Product Wheels Defined

A product wheel is a visual metaphor for a structured, regularly repeating sequence of the production of all of the materials to be made on a specific piece of equipment, within a reaction vessel, within a process system, or on an entire production line. The overall cycle time for the wheel is fixed. The time allocated to each product campaign (a "spoke" on the wheel—continuous operation on a single product) is relatively fixed, based on that product's average demand over the wheel cycle. The sequence of products is fixed, having been determined from an analysis of the path through all products which will result in the lowest total transition time or the lowest overall transition cost. Figure 13.1 depicts product wheel components.

The spokes can have different lengths, reflecting the different average demands of the various products: high demand products will have longer spokes (campaigns) than lower demand products.

Product wheels support a pull replenishment model. That is, the wheel will be designed based on average historical demand or on forecast demand for each

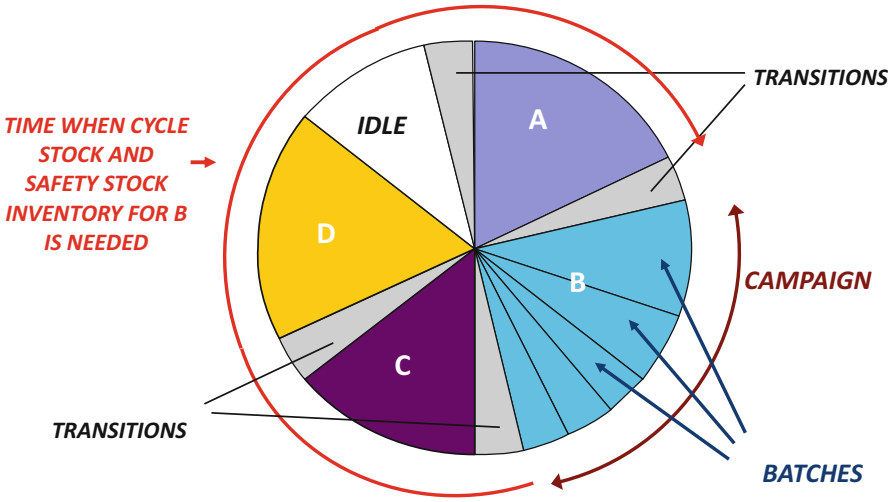


Fig. 13.1 Product wheel components

product, but what is actually produced on any spoke is just enough to replenish what has been consumed from the downstream inventory, in accordance with lean pull principles. Thus the size of each spoke can vary from cycle to cycle based on actual demand, but the total wheel cycle time will remain fixed.

13.2.1 Make to Stock, Safety Stock, and Make to Order

Product wheels are almost always used in "make to stock" operations. Only the cycle stock portion of the inventory is considered for the product wheel analysis. Yet we know that longer product wheels mean a longer "period of risk" and hence larger safety stocks (to maintain a given customer service level). However, safety stock is not included in the product wheel calculations discussed here because it grows relatively slowly with increases in product wheel length. (Safety stock usually increases only by the square root of the product wheel length—unlike cycle stock which is directly proportional to product wheel length.)

When only a small fraction of total demand is "make to order," this demand is sometimes assigned to a production unit along with the "make to stock" products. "Make to order" can be handled by putting "discretionary time" periods into the wheel design. The exact "make to order" products and amounts are not specified until near time for the period to begin. Planning for pre-set "make to order" production periods helps to define customer order lead time for those products, but total "make to order" time on the unit does need to be stable.

13.3 Product Wheels and the Economic Lot Scheduling Problem

The product wheel problem is an expression of the widely studied economic lot scheduling problem. Elmaghraby (1978) wrote a frequently cited review of the problem. Narro Lopez and Kingsman (1991) did another review, and variants of the problem continue to be studied (Teunter et al. 2008). Hsu (1983) showed that the problem is NP-hard.

The heuristic discussed here, simplifies the problem by assuming the user already has a good product sequence—and wants to keep it (or something close to it). Management wants to find the "sweet spot" balancing transition cost and inventory carrying costs, while maintaining capacity feasibility. The economic lot scheduling problem looks for that sweet spot. The heuristic discussed here provides a relatively simple approach—that has been implemented in many production areas.

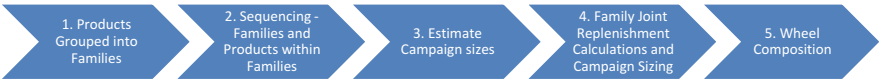


Fig. 13.2 Product wheel 5-STEP methodology

13.4 Product Wheel Methodology

The overall approach discussed, is summarized in the steps shown in Fig. 13.2. The rest of chapter will describe those steps in greater detail. This method for calculating product wheels has proven to be of significant value, having been implemented in manufacturing areas across a number of businesses.

13.4.1 Step 1: Product Families

When there are multiple products assigned to a production unit, they are frequently divided into groups—"families"—according to physical characteristics—size, chemical composition, etc. In a production context the goal is to subdivide the products assigned to a processing unit into families, based on common processing characteristics. All of the products in a family should be able to run sequentially, without requiring major transition time or cost (and the two frequently go together). Within a family product transitions are typically cheap and fast. Between families, however, product transitions are slower and/or more costly.

An example of a product family is products (e.g., polymers) using the same basic chemistry. When changing between products within the family, it may be possible to merely "plug flow" the next product and discard the mixed polymer coming through the line. However between families, it will be necessary to disassemble the production apparatus for a thorough cleanout.

Another example can be found with roll goods. Here a width change for the "mother roll" represents a significant effort, while changes in position of slitting knives, roll tension, packaging, etc., will be less difficult (faster and less costly). Products made from the same width mother roll would be in the same family.

When running product wheels correctly it will be a firm rule that families run in the product wheel sequence—and once a family is started, all of the products within the family that require product will be run. To run a different family's product out of order—within a different family (usually because it's expedited) is referred to as "breaking into the wheel." When intra-family transitions are expensive (time and/or cost) the decision to "break into a wheel" cannot be taken lightly. When a manufacturing line is running at close to capacity, breaking into the wheel may result in a downward spiral when one wheel break for expediting a product, leads to another wheel break, and so forth. Multiple wheel breaks use up line capacity

and customer delivery grows worse and worse because there's no longer sufficient capacity to service promised sales. (In a production area already running close to capacity, breaking into a wheel is sometimes compared to "breaking into jail"!)

13.4.2 Step 2: Sequence within the Wheel

The classic analysis of production sequencing might seem to start with looking at all of the products to be produced, developing a matrix of transition costs (or times—or both) and running a traveling salesman algorithm to develop the lowest cost (or minimum transition time) sequence. There are a number of reasons that optimized traveling salesman algorithms are not a priority with production planners—and are actually not done very often in manufacturing operations.

First of all, there may be hundreds of different products that could either be on order, or need replenishment for a make-to-stock system. Generation of a 100×100 matrix (even if just assigning arbitrarily large values to infeasible transitions) is not practical.

Fortunately, product families tend to reduce the problem to manageable (and often trivial) size. Since the bulk of transition expense and time is between families, the problem is often reduced to an "N × N" transition matrix where "N" is frequently in the range of just 3–5. It is also typical that some intra-family transitions are so expensive (or even physically impossible) that they can be immediately ruled out. Thus family sequence for product wheels is often followed for long periods of time (months or even a year or more), even though the product mix may vary over time and new products are introduced within existing families.

A second reason that a traveling salesman algorithm is not frequently used, is the quality of transition cost/time data itself. The precision of the transition data may not justify the rigor of optimization analysis! Even when the manufacturing operation collects transition data, it is likely to be filled with "special cause" and "one-off" situations where lessons are learned and problems overcome. Often "eyeballing the family transition data"—provides "good enough" sequences.

Within a family, transitions are usually much less expensive and time consuming—and the order is less important. Product wheels usually have a sequence within each family as well as a sequence of families. In any given wheel cycle, certain products are likely to be left out due to sales variation, but typically transition cost and time don't vary much within the family.

Besides the planner/scheduler, one of the best sources for determining a minimum cost and time sequence of product families is to ask experienced operators. They are usually the first to question poor sequencing decisions—because they are doing the work required to make "non-optimal" transitions!

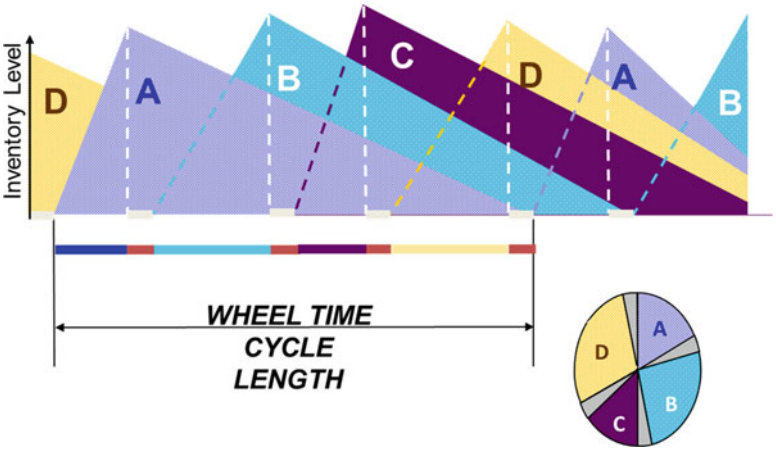


Fig. 13.3 Product wheel inventory plot

13.4.3 Step 3: Estimate Campaign Sizes

Campaign size (sometimes referred to as "Lot Size") is the quantity of a given product to be run during each spoke of the wheel. See Fig. 13.3. One of the goals of running product wheels is to run approximately the same campaign size on approximately the same time cycle. For instance if the wheel "turns" every 30 days, then enough will be produced to bring the cycle stock up to 30 days of sales (plus enough to replace any safety stock used during the cycle) approximately every 30 days.

The primary tradeoff in running product wheels is the balance between costs of carrying cycle stock versus transition costs. (Safety stock also increases with increasing wheel length—but much less than linearly—and is usually ignored in the calculations.) This inventory-transition tradeoff is the same tradeoff made with the well-known "Economic Order Quantity" ("EOQ") (Harris 1913). More applicable is an EOQ variant—the Economic Production Quantity (or Economic Production Lot) equation (Taft 1918) shown below:

$$\text{Economic Production Quantity} = \sqrt{\frac{2AD}{vr\,(1 - D/m)}}$$

where A, transition cost; D, demand rate in units/time period; v, unit cost of item ($/unit); r, fractional carrying cost of inventory per time period; m, production rate in units/time period.

Note that the D/m term goes to zero for purchased items, to give the form of the more familiar EOQ.

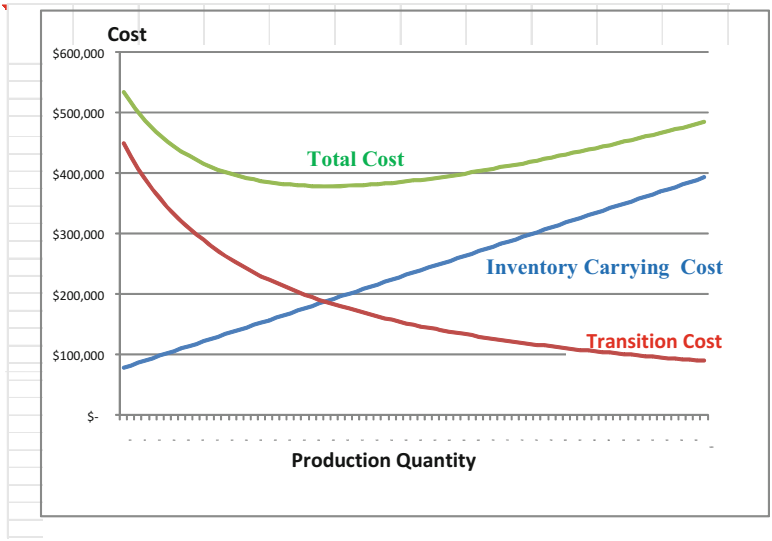


Fig. 13.4 Economic Order Quantity (EOQ) costs

The function is typically graphed as shown in Fig. 13.4.

The problem with using the "Economic Production Quantity" calculation above for product wheels is that each product is analyzed individually, whereas in a "wheel" the products run cyclically—*on the same basic cycle*. Nevertheless, the "Economic Production Quantity" gives at least a starting point for estimating favorable product campaign sizes (and hence campaign frequencies). (In the rare case that all products have the same production frequency—that frequency would be the answer on how often to "turn the wheel.")

13.4.4 Step 4: Product Families—"Joint Replenishment"

Edward Silver has written a number of very practical articles on coordinating "joint replenishment" in the context of economic production quantity. Specifically, he has published a "simple method" (Silver 1976) for calculating how often to produce a family of products (product family cycle length) and which products to produce every time the family is produced, every second time, every third time, . . . (product frequency in family). This method (sometimes referred to as "EOQ Family") is getting closer to solving the product wheel problem, but still does not directly solve the problem. The product family cycle length is very likely to be different between families. If the product family cycle lengths happen to be integer multiples of one another, then the families might possibly be arranged in "product wheel cycles." However even this solution still may not be feasible. If some product families don't

run every base cycle, or the product frequency in family is not "1" for every product, the cycles are likely to require different processing times and some cycles could exceed the base cycle length.

13.4.5 Using Silver's Joint Replenishment Heuristic to Determine "Product Wheel Length"

13.4.5.1 Example Joint Replenishment Solution

The following example shows an implementation of Silver's Joint Replenishment heuristic (sometimes referred to as "EOQ Family") implemented in a spreadsheet (Tables 13.1 and 13.2). (This implementation uses visual basic calculations as part of the spreadsheet analysis.)

We define *Product Wheel Length* as the target time for cycling through the wheel. We define *Campaign Cycle* as the time between starting cycles (campaigns) of an individual product. Thus the *Campaign Cycle* for every product will be equal to an integer multiple (usually 1) of *Product Wheel Length*. The Joint Replenishment solution suggests limits for an estimate of optimal *Product Wheel Length* in the "EOQ Family: Product Family Cycle Days" column—looking (only) at those products that are produced every turn of the wheel (=1 in the "EOQ Family: Frequency . . . " column). (Commercial factors such as shelf life and storage capacity also limit *Product Wheel Length*.) For the example in Tables 13.2 and 13.3, we would look at using *Product Wheel Lengths* in the range of 19–42 days.

It is straightforward to extend the Joint Replenishment Spreadsheet to perform product wheel cost analysis (looking at cost of transitions plus cost of carrying cycle stock). With that cost model we can plug in different values of *Product Wheel Length* and quickly evaluate a number of alternatives in the range.

We use the following rules for selecting *Campaign Cycles* for each product.

Define (based on Joint Replenishment results):

- *Product Frequency* = Value in column "EOQ Family: Product Frequency . . ."
- *Product Cycle* = Value in "EOQ Family: Total Days . . ." for that product
- *Family Cycle* = Value of *Campaign Cycle* for those product s in the family with a *Product Frequency* = 1

Calculate the *Campaign Cycle* (days between campaigns) for each product, as shown in the three steps below. Note that *Campaign Cycle* will be an integer multiple of *Product Wheel Length*. Also every product in a family will be an integer multiple of one another. Calculate *Campaign Cycle* first for products having *Product Frequency* =1, so that a *Family Cycle* value can be determined for each family.

1. If *Product Cycle* <= *Product Wheel Length*, set *Campaign Cycle* = *Product Wheel Length*

Table 13.1 Input data

Product name	Family number	Forecast annual demand	Cost per unit	Family setup cost	SKU setup cost	Days setup time for family	Days setup time for SKU	Inventory carrying costs	Days available for production
Prod1	1	459,281	$32.78	$5184	$91	0.02		25 %	335
Prod2	2	310,254	$22.48	$9500	$125	0.04		25 %	335
Prod3	2	247,557	$22.77	–	$18,200	0.04		25 %	335
Prod4	2	403,498	$21.16	–	$240	0.02		25 %	335
Prod5	2	122,291	$23.59		$95	0.02		25 %	335
Prod6	2	122,291	$22.1		$86		0.001	25 %	335
Prod7	2	161,000	$28.25		$200	0.02		25 %	335
Prod8	2	294,917	$25.44	–	$1250	0.02		25 %	335
Prod9	2	361,843	$27.42	–	$500			25 %	335
Prod10	3	67,206	$28.83	$30,368	$500	1.00		25 %	335
Prod11	3	502,058	$27.02	–	$500			25 %	335
Prod12	3	32,300	$27.99		$15,000			25 %	335

Table 13.2 Joint Replenishment results from Table 13.1 input

Product name	Family number	EOQ family: Campaign size	EOQ family: Product frequency in family (1 = every cycle, 2 = every other cycle, ...)	EOQ family: Total days between product campaigns
Prod1	1	24,316	1	17.7
Prod2	2	17,290	1	18.7
Prod3	2	27,591	2	37.3
Prod4	2	22,486	1	18.7
Prod5	2	6815	1	18.7
Prod6	2	6815	1	18.7
Prod7	2	8972	1	18.7
Prod8	2	16,435	1	18.7
Prod9	2	20,165	1	18.7
Prod10	3	8494	1	42.3
Prod11	3	63,452	1	42.3
Prod12	3	12,247	3	127.0

2. If *Product Frequency* = 1 and *Product Cycle* > *Product Wheel Length*, then set the *Campaign Cycle* to the nearest integer multiple of the *Product Wheel Length*
3. If *Product Frequency* > 1, then set the *Campaign Cycle* = *Family Cycle x Product Frequency*

Table 13.3 shows annual cost calculations for three candidate values of *Product Wheel Length*: 21, 28, and 35. The column "Wheel to use for annual cost evaluation: Days between product production" is the *Campaign Cycle* for each product, calculated using the steps above. The user would in like manner try a variety of *Product Wheel Length*s to find one that produces a low total annual cost and works within the limitations of the production process (for instance minimum campaign size).

13.4.5.2 Capacity Versus *Product Wheel Length*

One of the first results to check with this method is to ask if there's sufficient capacity. Very low transition costs will drive the heuristic to produce very short campaigns—potentially taking up significant production unit time in transition. Increasing the *Product Wheel Length* reduces utilization—but at the cost of greater inventory carrying cost. Typically when using the heuristic described, one calculates capacity as part of the spreadsheet calculation. If there's not sufficient capacity, then increase the input *Product Wheel Length* and recalculate the *Campaign Cycles* for the wheel.

Table 13.3 Total annual cost and capacity calculations

Product name	Family number	EOQ family: Campaign size	EOQ family: Product frequency in family (1 = every cycle, 2 = every other cycle, ...)	EOQ family: Total days between product campaigns	Product wheel length = **21**		Product wheel length = **28**		Product wheel length = **35**	
					Wheel to use for annual cost evaluation: Days between product production		Wheel to use for annual cost evaluation: Days between product production		Wheel to use for annual cost evaluation: Days between product production	
Prod1	1	24,316	1	17.7	21.0		28.0		35.0	
Prod2	2	17,290	1	18.7	21.0		28.0		35.0	
Prod3	2	27,591	2	37.3	42.0	Annual demand	56.0	Annual demand	70.0	Annual demand
Prod4	2	22,486	1	18.7	21.0	3,084,496	28.0	3,084,496	35.0	3,084,496
Prod5	2	6815	1	18.7	21.0		28.0		35.0	
Prod6	2	6815	1	18.7	21.0	Annual capacity	28.0	Annual capacity	35.0	Annual capacity
Prod7	2	8972	1	18.7	21.0	7,473,880	28.0	7,625,321	35.0	7,452,726
Prod8	2	16,435	1	18.7	21.0		28.0		35.0	
Prod9	2	20,165	1	18.7	21.0		28.0		35.0	
Prod10	3	8494	1	42.3	42.0	Total annual cost	56.0	Total annual cost	35.0	Total annual cost
Prod11	3	63,452	1	42.3	42.0	$ 1,506,428	56.0	$ 1,556,833	35.0	$ 1,665,156
Prod12	3	12,247	3	127.0	126.0		168.0		105.0	

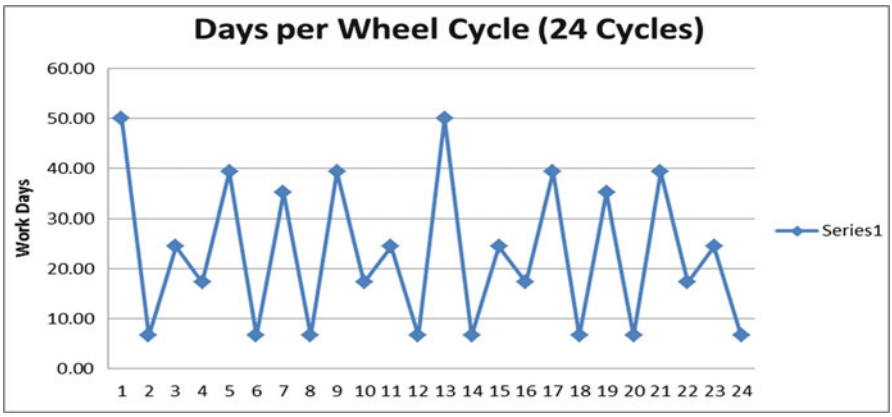


Fig. 13.5 Showing the days required to run each cycle of the wheel described in Table 13.4. Product wheel cycles are on the *X*-axis

13.4.6 Step 5: Wheel Composition

The principle that Silver's algorithm uses—production of a product family will cost less when we run low demand products less frequently than high demand products—makes common sense. However we see from the results in Table 13.2, that "Prod3" should run about every second product wheel cycle and Prod12 every third wheel cycle to achieve minimum total cost (transition cost plus inventory carrying cost). When there are multiple products that do not run every wheel cycle, it raises the problem of how to "balance" successive wheel cycles.

As we've seen, a production unit may be assigned some products that "optimally" run every "turn of the wheel" (*Campaign Cycle* = *Product Wheel Cycle*), others that should run only half as frequently as the "big sellers," and still other very low volume products and product families that should run only every third or fourth cycle. The issue now is which product families should run together in what cycles. Should the "every second cycle" products run on the first and third wheel cycles—or second and fourth? This is the same as asking—"which of those families should start on the first cycle and which on the second?" Note that all products in a *family* need to be coordinated to run on either the same cycles, or for low volume products, on a subset of those cycles—in order to minimize family transition costs.

The potential problem (that is certainly seen in real life) is that the wheels can end up very "unbalanced"—with some product wheel cycles ("turns of the wheel") requiring much more time than other cycles. For instance a nominal 25-day product wheel could have wide swings in duration from cycle to cycle—requiring the wheel's nominal 25 days of cycle stock to last for 50 days or more on some cycles.

The following is an example of a product wheel design made with all families starting in cycle 1 (default solution). (A larger example problem is used here,

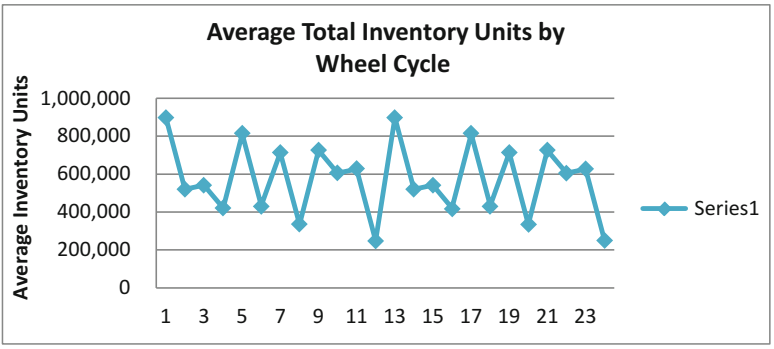


Fig. 13.6 Showing the inventory swings produced by the product wheel plan in Table 13.4. Product wheel cycles are on the *X*-axis

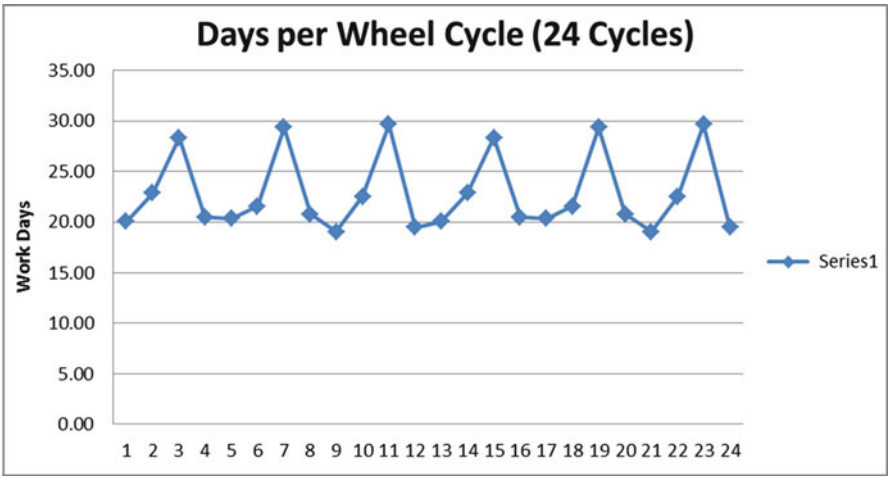


Fig. 13.7 Showing the days required to run each cycle of the wheel described in Table 13.5. Product wheel cycles are on the *X*-axis

to illustrate the problem more clearly.) Cycle durations range from 8–50 days. Obviously total inventory makes similar swings (with some products likely to run out) (Figs. 13.5, 13.6, and Table 13.4).

With judicious selection of starting cycles (by observing what frequencies that families run, and putting families into different starting cycles), the extremes of cycle durations can be avoided (Figs. 13.7, 13.8, and Table 13.5).

The improvement in wheel composition came from balancing the family days of production among the starting cycles. (When there are more products, it's usually easier to get better balance.) The rules for assigning products to starting cycles are:

Table 13.4 Produced from a spreadsheet program to "cost out" different wheel lengths

Product name	Wheel to start production on: first wheel (=1), second wheel (=2), ...	Produce every wheel (=1), every other wheel (=2), ...	Avg. days production during campaign	Family setup (0 or 1)	Family setup time (Days)	SKU setup (Days)	Safety stock	Demand rate (Units/Day)	Days per campaign	Family number
P1	1	2	3.53	1	0.02		2103	1258	3.55	1
P2	1	3	3.72	1	0.04		1095	850	3.76	2
P3	1	3	2.97	1	0.04		1245	678	3.02	3
P4	1	2	2.99	1	0.02		1847	1105	3.01	4
P5	1	3	1.35	1	0.02		615	335	1.37	5
P6	1	3	1.41			0.00	615	335	1.41	5
P7	1	2	2.18	1	0.02		737	441	2.20	6
P8	1	2	2.17	1	0.02		1350	808	2.19	7
P9	1	2	2.66				1656	991	2.66	7
P10	1	1	0.69	1		0.02	780	524	2.69	8
P11	1	4	2.05	1	0.02		746	376	2.07	9
Discretionary	1	2	1.15	1	0.01			0	1.16	10
P12	1	2	3.07	1	0.02		1826	1093	3.09	11
P13	1	4	1.68	1	0.02		560	282	1.70	12
P14	1	1	3.98	1	0.02		4319	2901	4.00	13
P15	1	3	1.04	1	0.02		467	255	1.06	14
P16	1	4	1.03	1	0.02		366	184	1.05	15
P17	1	4	7.65				2731	1376	7.65	15
P18	1	4	2.34				1382	420	2.34	15

Note that the "Starting Cycle" in column 2, defaults to "1" for all products

Table 13.5 Same output from a product wheel "costing" spreadsheet as Table 13.4

Product name	Wheel to start production on: first wheel (=1), second wheel (=2), ...	Produce every wheel (=1), every other wheel (=2), ...	Avg. Days production during campaign	Family setup (0 or 1)	Family setup time (Days)	SKU setup (Days)	Safety stock	Demand rate (Units/Day)	Days per campaign	Family number
P1	2	2	3.53	1	0.02		2103	1258	3.55	1
P2	1	3	3.72	1	0.04		1095	850	3.76	2
P3	2	3	2.97	1	0.04		1245	678	3.02	3
P4	1	2	2.99	1	0.02		1847	1105	3.01	4
P5	3	3	1.35	1	0.02		615	335	1.37	5
P6	3	3	1.41			0.00	615	335	1.41	5
P7	2	2	2.18	1	0.02		737	441	2.20	6
P8	1	2	2.17	1	0.02		1350	808	2.19	7
P9	1	2	2.66				1656	991	2.66	7
P10	1	1	0.69	1		0.02	780	524	2.69	8
P11	2	4	2.05	1	0		746	376	2.07	9
Discretionary	2	2	1.15	1	0			0	1.16	10
P12	2	2	3.07	1	0		1826	1093	3.09	11
P13	1	4	1.68	1	0		560	282	1.70	12
P14	1	1	3.98	1	0		4319	2901	4.00	13
P15	2	3	1.04	1	0		467	255	1.06	14
P16	3	4	1.03	1	0		366	184	1.05	15
P17	3	4	7.65				2731	1376	7.65	15
P18	3	4	2.34				1382	420	2.34	15

But column 2 has been adjusted to smooth product wheel cycle length

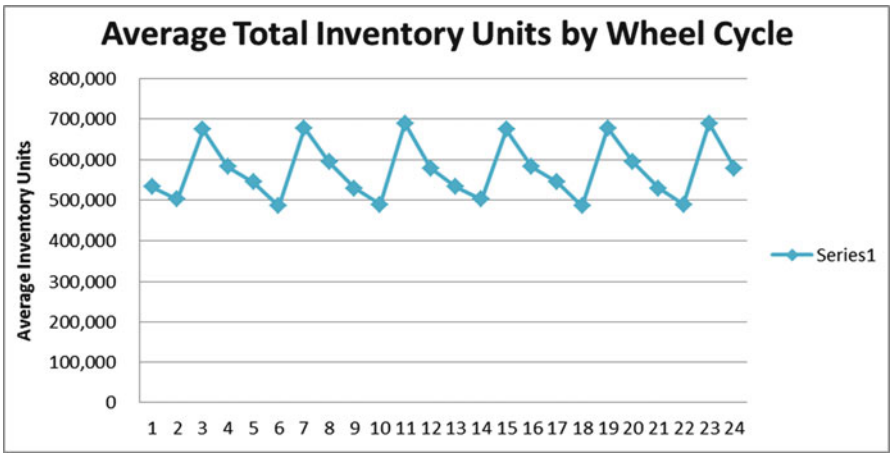


Fig. 13.8 Showing the inventory swings produced by the product wheel plan in Table 13.5. Product wheel cycles are on the *X*-axis

1. To minimize family transition costs, higher volume products in a family (run every time the family runs) should begin in the same cycle. Low volume family products (that run less frequently) should run on a subset of the cycles where the high volume products of the same family run.
2. Products cannot begin in a cycle that exceeds their frequency. (For instance products made every other wheel can only start in cycles 1 or 2—not 3. Products made every third wheel can only start in cycles 1, 2, or 3. . .)

Excel graphical analysis like that above shows where the problem is—and allows testing different solutions. However, even a small problem like the one above can be difficult to minimize by trial and error.

13.5 Data Considerations

13.5.1 Transition Time and Cost

Correct transition cost data is critical to any EOQ related method. Costs for yield loss (extra raw materials, energy, and waste disposal) and maintenance materials, are usually readily available. Manufacturing labor is usually *not* charged to transitions (unless overtime is required or labor is truly a "variable cost" in the short term). The most frequent (and largest) error in transition cost calculation is when the analyst charges the opportunity cost of time on the production line—when the business is not "oversold." (If there's not a buyer for the extra product that would have been made in place of stopping for the transition, then it's obviously *not* allowable to

charge for the time as a cost of the transition.) If the market for products made on the line does become oversold, then the transition cost should include the opportunity cost of the transition time (margin on the product that could have been produced in place of the transition)—which usually drives the transition cost much higher than before.

Transition time and cost obviously relate to both the product being transitioned "from," and the product being transitioned "to." Product wheel methodology is to run the same sequence repetitively. However, it is possible that some products may only run in every second or third "turn" of a wheel—and thus a given product may not always be preceded by the same product. This product wheel heuristic depends on the user to determine when the transition cost and time used for calculation is not appropriate for the actual order in the wheel. In these cases the user should adjust the transition cost and time and repeat the analysis. (Changes in transition cost could impact the prescribed wheel length. Changes in transition time only impact capacity feasibility.)

13.5.2 Demand and Uptime Variability

Product wheel planning can be maintained in the face of demand surges by: (1) planning for "slack time" in the wheel and (2) allowing the wheel to run a little longer than planned (using safety stock to cover sales and shortening subsequent wheel "turns" when possible). Production planners are warned to be aware of substantial shifts upward in average demand—and to recalculate wheel lengths (and offload production to other facilities) when such a shift occurs.

Uptime losses are typically separated into two classes: "short" routine outages, and much longer (sometimes catastrophic) outages that occur with much lower frequency. Short, routine outages are counted as "normal" downtime and production rates are factored down to account for these. Some lower frequency outages may be covered by safety stock when they are not routine. However very low frequency, long duration "catastrophic" outages are more often responded to *ad hoc* (and may be grounds for declaring a *Force Majeure* and serving customers accordingly).

13.6 Revising Product Wheels

Industrial data, particularly demand data, is never static. Developing and revising product wheels is an ongoing effort. Spreadsheet calculations and particularly Silver's Joint Replenishment heuristic can be built into spreadsheet tools and offer a rapid means for approximate solutions that are usually as "good as the data." Product wheels should be recalculated whenever:

1. Total demand changes by 10–20 %—or changes enough that the current wheels need to be lengthened to keep from running short of capacity. Note that during a given planning period, increased demand for individual products will often be balanced by decreased demand for others—leaving the product wheel cost effective and feasible.
2. Products are shifted between production units (usually in response to demand changes).
3. Transition times and/or costs change significantly.
4. Process efficiencies significantly reduce processing times.

Product wheel composition remains a challenge, and offers opportunity for improvement over current manual analysis.

13.7 Summary

Prior to introducing product wheels into their scheduling logic, many operations tended to treat each day's or each week's production plan as a totally new thing, to be scheduled from scratch. They would try and follow the ideal transition sequence, but often found themselves forced to "break in" to the sequence when a particular product ran out. With wheels, they quickly learn that all of the routine products are pre-planned to cover expected demand, so they can focus their attention on the few unique situations that require special attention. And they now have enough mental bandwidth to deal with these abnormal situations and crises appropriately.

A number of innovative companies have employed product wheels to great advantage: DuPont (chemicals, paints, sheet goods, extruded polymers), Dow (chemicals), AstraZeneca (pharmaceuticals), ExxonMobil (oil and gas), and APPVION (paper products). The sidebar shows the benefits they have found from wheel scheduling. As a specific example Dow Chemical typically sees inventory reductions of 10–20 %, 10–25 % higher customer fill rates, 30–40 % shorter lead times, and greater predictability and stability. The latter benefits are what some users appreciate the most, the dramatic reduction in the "noise level" in the system, the fact that scheduling chaos has been replaced by stability, predictability, and fixed patterns which allow everyone to get into a production rhythm.

13.8 Acknowledgement

The authors wish to thank Dr. Rami Musa, DuPont Co. for valuable suggestions to this chapter.

Some of the material in the *Introduction*, *Product Wheels Defined*, and *Summary* sections was adapted with permission from *The Product Wheel* Handbook—*Creating Balanced Flow in Multi-Product Process Operations*. Productivity Press, 2013.

References

Altamirano Rua T (2011) Migration, remittances and development in times of crisis. Editorial Fund/Pontiphical Catholic University, Peru

Elmaghraby SE (1978) The economic lot scheduling problem (ELSP) review and extension. Manage Sci 24(6):1978

Harris FW (1913) How many parts to make at once. Factory Mag Manage 10(2):135–136, 152. Also reprinted in Operations Research 38(6):947–950, 1990

Hsu W-L (1983) On the general feasibility test of scheduling lot sizes for several products on one machine. Manage Sci 22(1):93–105

King PL, King JS (2013) The product wheel handbook—creating balanced flow in multi-product process operations. Productivity Press, Boca Raton

Narro Lopez MA, Kingsman BG (1991) The economic lot scheduling problem: theory and practice. Int J Prod Econ 23:147–164

Silver EA (1976) A simple method of determining order quantities in joint replenishments under deterministic demand. Inst Manage Sci 22(12):1351–1361

Taft EW (1918) Formulas for exact and approximate evaluation handling cost of jigs and interest charges of product manufactured included. Iron Age 101(1410):1412

Teunter R, Konstantinos K, Tang O (2008) Multi-product economic lot scheduling problem with separate production lines for manufacturing and remanufacturing. Eur J Oper Res 191: 1241–1253

Womack JP, Jones DT (1996) Lean thinking. Free Press, New York

Index

G. Rabadi (ed.), *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, International Series in Operations Research & Management Science 236, DOI 10.1007/978-3-319-26024-2