THE DYNAMIC, RESOURCE-CONSTRAINED SHORTEST PATH PROBLEM ON AN ACYCLIC GRAPH WITH APPLICATION IN COLUMN GENERATION AND A LITERATURE REVIEW ON SEQUENCE-DEPENDENT SCHEDULING

A Dissertation

by

XIAOYAN ZHU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2005

Major Subject: Industrial Engineering

THE DYNAMIC, RESOURCE-CONSTRAINED SHORTEST PATH PROBLEM ON AN ACYCLIC GRAPH WITH APPLICATION IN COLUMN GENERATION AND A LITERATURE REVIEW ON SEQUENCE-DEPENDENT SCHEDULING

A Dissertation

by

XIAOYAN ZHU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Wilbert E. Wilhelm
Committee Members,	Illya V. Hicks
	Sergiy Butenko
	David Larson
Head of Department,	Brett A. Peters

December 2005

Major Subject: Industrial Engineering

ABSTRACT

The Dynamic, Resource-Constrained Shortest-Path Problem

on an Acyclic Graph with Application in Column Generation and

a Literature Review on Sequence-Dependent Scheduling. (December 2005)

Xiaoyan Zhu, B.En., Tsinghua University;

M.S., MIT & Nanyang Technological University;

M.S., Texas A&M University

Chair of Advisory Committee: Dr. Wilbert E. Wilhelm

This dissertation discusses two independent topics: a resource-constrained shortest-path problem (RCSP) and a literature review on scheduling problems involving sequence-dependent setup (SDS) times (costs).

RCSP is often used as a subproblem in column generation because it can be used to solve many practical problems. This dissertation studies RCSP with multiple resource constraints on an acyclic graph, because many applications involve this configuration, especially in column genetation formulations. In particular, this research focuses on a dynamic RCSP since, as a subproblem in column generation, objective function coefficients are updated using new values of dual variables at each iteration. This dissertation proposes a pseudo-polynomial solution method for solving the dynamic RCSP by exploiting the special structure of an acyclic graph with the goal of effectively reoptimizing RCSP in the context of column generation. This method uses a one-time "preliminary" phase to transform RCSP into an unconstrained shortest path problem (SPP) and then solves the resulting SPP after new values of dual variables are used to update objective function coefficients (i.e., reduced costs) at each iteration. Network reduction techniques are considered to remove some nodes and/or arcs permanently in the

preliminary phase. Specified techniques are explored to reoptimize when only several coefficients change and for dealing with forbidden and prescribed arcs in the context of a column generation/branch-and-bound approach. As a benchmark method, a label-setting algorithm is also proposed. Computational tests are designed to show the effectiveness of the proposed algorithms and procedures.

This dissertation also gives a literature review related to the class of scheduling problems that involve SDS times (costs), an important consideration in many practical applications. It focuses on papers published within the last decade, addressing a variety of machine configurations - single machine, parallel machine, flow shop, and job shop - reviewing both optimizing and heuristic solution methods in each category. Since lot-sizing is so intimately related to scheduling, this dissertation reviews work that integrates these issues in relationship to each configuration. This dissertation provides a perspective of this line of research, gives conclusions, and discusses fertile research opportunities posed by this class of scheduling problems.

To my dear sister Xiaona Zhu

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Dr. Wilbert E. Wilhelm, Department of Industrial Engineering, Texas A&M University, for his continuous guidance and many educational contributions. Throughout my Ph.D. program, I have learned so many aspects of life from him. Without his guidence and encouragement I could not have finished my Ph.D. studies smoothly.

Dr. Wilhelm has co-authored papers that are in review and I am very grateful for his significent contributions and valuable hard work. I also would like to thank Dr. Wilhelm for his helpful comments and suggestions that have improved both the contents and the structure of the dissertation.

I wish to thank Dr. Illya V. Hicks, Dr. Sergiy Butenko and Dr. David Larson for serving on the committee and for giving valuable suggestions. Thanks also to my peers: Dong Liang, Deepak Warrier and Brijesh Rao for the wonderful time and discussions we shared together. Most importantly, I would like to thank my husband, Tao Yuan, for his encouragement and aid in solving some computer problems.

TABLE OF CONTENTS

CHAPTER Page

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

This dissertation addresses two independent topics: a resource-constrained shortest-path problem (RCSP) and a literature review related to scheduling problems that involve sequence-dependent setup (SDS) times (costs).

The first topic studied in this dissertation is a dynamic RCSP, which is a variant of the classical single-source, single-sink shortest-path problem (SPP). Consider a directed graph (digraph) in which each arc has an associated cost and a vector representing the resources required to traverse it. Then, RCSP is to find a shortest path (i.e., the path with the least total arc cost) from the source node to the sink node with a total consumption of each type of resource that observes a given upper bound. We denote RCSP with a single resource constraint as SRCSP and RCSP with multiple resource constraints as MRCSP. RCSP is often used as a subproblem in column generation (CG); many applications (e.g., in scheduling) may be modeled using an acyclic graph and multiple resource restrictions. Thus, this dissertation studies RCSP with multiple resource constraints on an acyclic graph in the context of CG. When RCSP is used as a subproblem in CG, objective function coefficients are updated using new values of dual variables at each CG iteration and, consequently, RCSP must be reoptimized with respect to these new objective function coefficients. This renders the dynamic property of RCSP. Moreover, numerous mixed integer linear programming problems (MIPs) have been solved successfully using branch-and-bound (B&B), incorporating CG at each node of the B&B tree to compute (tight) bounds. When RCSP is used as a subproblem in CG/B&B, some arcs in the graph may be forbidden or prescribed (i.e., with associated decision variables fixed to 0 or 1,

---

This dissertation follows the style and format of *IIE Transactions*.

respectively) by the branching rule. Thus, an effective solution method for RCSP must offer the ability to handle repeated reoptimization, subject to fixed (i.e., forbidden and prescribed) arcs in the context of CG/B&B. Currently available methods do not satisfy these needs, so this dissertation aims to fill this void.

Another topic studied in this dissertation is a class of scheduling problems that involve SDS times (costs). In general, setup includes work required to prepare a machine (or process) to produce parts of a given type, including setting jigs and fixtures, adjusting tools, and provisioning material. Because of their prevalence in, and importance to, industry and because of the challenges they present to solution methodologies, scheduling problems that involve SDS have attracted the interests of many researchers. This dissertation contributes by focusing on recent results and providing a technical perspective of the topic.

The dissertation is structured in two parts; each corresponds to one of the two topics. Part I, consisting of Chapters II-IX, investigates RCSP. Part II, consisting of Chapters X-XV, reviews the literature on scheduling problems with SDS. The last chapter presents conclusions of this dissertation research and outlines fertile opportunities for future work.

CHAPTER II

INTRODUCTION TO RCSP

This chapter presents the scope of RCSP (Section 2.1), addressing various types of constrained SPPs (CSPs); motivations of this research (Section 2.2), focusing on the applications of RCSP; objectives of the research on RCSP (Section 2.3); and the organization of Part I (Section 2.4).

2.1. Scope

The classical SPP is to find a path with minimum cost from a given source node to a given sink node on a digraph. A SPP subject to different types of constraints results in distinct types of CSPs. In general, a CSP is to find a minimum cost path with respect to objective function coefficients while satisfying a set of constraints, which may impose some additional characteristics on the path in a graph. The term "cost" in Part I of this dissertation refers to the objective function coefficients in SPP or CSP, although it may represent distance, time or another measure, depending on the application.

CSP is important because, by incorporating different constraints in SPP, CSP can be used to model and solve a variety of practical problems. The types of constraints most frequently considered include time-window constraints for nodes (Desaulniers et al. (1997), Desrochers et al. (1992)) or arcs (Jaumard et al. (1996, 1998)); traffic-light constraints (Chen and Yang (2000)); on-off time-switch constraints (Chen and Yang (2003)); time-schedule constraints (Chen and Tang (1997)) in which departures from nodes are only allowed at some discrete time points; hop constraints (Daul and Gouveia (2004)), which require a feasible path with at most a given number of hops (arcs); equity constraints (Gopalan (1990)); label-constraints (Barrett et al. (2000), Sherali et al. (2003)); bottleneck constraints (Berman et al.

(1990), Shin et al. (1995), Wilhelm (1999)), which limit the resource requirement (single resource) on each arc in the shortest path to be no greater than a given threshold; constraints from a set of forbidden paths that can not be part of any feasible solution (Villeneuve and Desaulniers (2005)); constraints from a set of prescribed arcs that are required to be part of the shortest path (Chen and Hung (1994)); as well as budget- or resource-constraints (Holmberg and Yuan (2003), Elimam and Kohler (1997)).

This dissertation focuses on RCSP with single or multiple knapsack-type resource limitations; for example, a resource can be time, distance, capacity, money, workload, or reliability requirement. Given a digraph in which each arc has an associated cost and a vector representing the resource required to traverse it, RCSP can be depicted as finding a shortest path from a source node to a sink node with a total consumption of each type of resource that observes a given upper bound. The next chapter presents a formal description of RCSP.

The shortest path problem with time windows (SPPTW), or, more generally, resource windows (SPPRW), is another type of CSP, which is to find a shortest path between a given pair of source and sink nodes such that the cumulative requirement of each type of resource at each node on the shortest path lies within a given resource window associated with that node. Analogous to a time window, a resource window at a node defines the smallest and largest amounts of resource that the cumulative resource requirement along a path from the source node to that node can be. In this dissertation, the term "SPPTW" represents the SPP with window constraints on a single resource (usually time), and the term "SPPRW" represents the general case (i.e., SPP with multiple types of resource-window constraints). Resource windows can be classified into two types: "hard" or "soft". In the former case, if one or more resource-window constraints are not satisfied, then the solution becomes infeasible (Kolen et al. (1987), Russell (1995), Bramel and Simchilevi (1996)). In the latter case, a cost penalty is incurred if the

cumulative resource requirement at a node is outside its resource window (Balakrishnan (1993)) (the penalty is typically assumed to be a linear function of the amount of violation). For example, if the resource is time, as in a vehicle routing problem with time windows (VRPTW), a hard time window requires a vehicle that arrives at a customer too early to wait to service the customer within the window, and that the vehicle can not arrive later than the due date (i.e., the upper bound of the window). The SPPRW most extensively investigated is the one with hard resource windows. Apparently, RCSP can be regarded as a SPPRW with a hard window at each node for each resource that ranges from 0 to the total resource limitation. However, Chapter V shows that RCSP on an acyclic graph can be transformed in polynomial time into SPPRW with much tighter resource windows. The next chapter presents a formal description of SPPRW.

2.2. Motivation

RCSP finds application in many areas, including transportation and communication (Holmberg and Yuan (2003)), network routing (Desrochers et al. (1992)), wastewater treatment (Elimam and Kohler (1997)), global supply chain design (Wilhelm et al. (2005b)), investment planning and project evaluation (Bard and Miller (1989)), transfer-line balancing (Dolgui et al. (2004)), scheduling and planning (Avella et al. (2004); Mingozzi et al. (1999)), as well as in some classical combinatorial problems, including generalized assignment, matching, and traveling salesman problems (Houck et al. (1980)). Avella et al. (2004) discussed the applications of RCSP in the management and control of a vehicle fleet on a road network.

In an important class of applications, RCSP appears as a subproblem in CG, which has been successfully used to solve some well-known problems like vehicle routing (Desrochers et al. (1992); Desrosiers et al. (1988); Desrosiers et al. (1984)), crew scheduling (Mingozzi et al. (1999); Desrochers and Soumis (1989); Lavoie et al. (1988)), prescribing the content and timing

of products upgrades (Wilhelm et al. (2003)), optimizing placing and picking operations on dual-head placement machines (Wilhelm et al. (2004) and Wilhelm et al. (2005a)), and multicommodity flow problems (Holmberg and Yuan (2003)). CG was first suggested by Ford and Fulkerson (1958) for a problem involving maximizing flow in a multicommodity network. It was then presented by Dantzig and Wolfe (1960) for linear programs with decomposable structures and by Gilmore and Gomory (1961) for the cutting-stock problem. A good recent survey on CG was provided by Wilhelm (2001). CG requires a model to be decomposed into two parts: the master problem and the subproblem(s). RCSP is one of most important subproblems in many large-scale real-world problems because it can be used to model the complex logic by which many systems operate (Wilhelm (2001)). Modern interest in CG was stimulated by the work in the context of VRPTW (e.g., Desrosiers et al. (1984); Desrochers et al. (1992); Desrosiers et al. (1993)). Desrochers et al. (1992) solved VRPTW using a B&B approach for the integer set-partitioning formulation of VRPTW. They solved the linear relaxation of this set-partitioning formulation by CG, generating feasible columns by solving SRCSP with additional time-window constraints using dynamic programming (DP). Recently, Holmberg and Yuan (2003) solved a multicommodity network-flow problem with side constraints, using CG with SRCSP as a subproblem for each commodity to deal with side constraints. The problem arose in the telecommunications area and the side constraints may represent the time-delay or reliability requirements on paths that are used for routing.

In particular, many applications can be represented on an acyclic graph, e.g., nurse scheduling (Jaumard et al. (1998)), crew pairing (Desaulniers et al. (1997)), crew scheduling (Mingozzi et al. (1999)), assembly system design with tool changes (Wilhelm (1999)), prescribing the content and timing of products upgrades (Wilhelm et al. (2003)), transfer line balancing (Dolgui et al. (2004)), and simultaneous operational flight and pilot scheduling

(Stojković and Soumis (2001)). Typically, applications in the area of scheduling can be modeled on an acyclic graph (van den Akker et al. (2000)).

These types of applications, especially the class for which RCSP is a subproblem in CG, motivate this dissertation research. This dissertation contributes by proposing a new, improved approach for repeatedly solving RCSP on an acyclic graph in CG/B&B. Consequently, it can speed up CG approaches for problems that can be solved using CG with RCSP as a subproblem. Specialized algorithms handle issues related to preprocessing, reoptimization, and fixed arcs in CG/B&B.

2.3. Objectives

The objectives of this dissertation research on RCSP are: i) a review of the literature that relates applications and existing algorithms for RCSP; ii) an effective solution method for the dynamic RCSP, including preprocessing techniques, reoptimization methods, and procedures for dealing with fixed arcs; iii) some extensions of the proposed method; and iv) computational evaluation.

2.4. Organization of part I

The remainder of part I is structured as follows. Chapter III reviews related literature and Chapter IV gives the formal descriptions of RCSP and SPPTW and introduces the main acronyms that are used in Part I of this dissertation. Chapters V and VI propose new algorithms for solving RCSP. Chapter VII presents a computational evaluation to show the effectiveness of the proposed algorithms. Chapter VIII investigates the special issues that arise when the new method is applied in the context of CG and CG/B&B. Finally, Chapter IX provides some extensions of the proposed algorithms.

CHAPTER III

LITERATURE REVIEW: RCSP

This chapter presents a comprehensive literature review. Starting from an overview of SPP in Section 3.1, consecutive sections overview different aspects of RCSP, including complexity (Section 3.2), available solution algorithms (Section 3.3), preprocessing techniques (e.g., network reduction and resource-window tightening) (Section 3.4), reoptimization (Section 3.5), and issues related to fixed arcs (Section 3.6). Finally, Section 3.7 summaries this entire chapter, emphasizing the necessity of this dissertation research.

3.1. SPP

The unconstrained, single-source, single-sink SPP is concerned with finding a minimum cost through a digraph path from a source node to a sink node. SPP is important because it finds numerous applications and generalizations, for example, in communications and transportation networks (Deo and Pang (1984)), design of quality control systems (White (1969)), scheduling (Gamache et al. (2005)), and many other areas. SPP is often used as a subproblem in CG approaches, for example, in the linear multicommodity flow problem (Babonneau et al. (2004)), vehicle routing problem (Christofides et al. (1981)), and single-machine scheduling problem (van den Akker et al. (2000)). Glover et al. (1985a) described additional applications. SPP is solvable in polynomial time and has been widely investigated. Several solution methods have proven to perform very well in practice, including the label-setting algorithm of Dijkstra (1959) for nonnegative arc costs, and the label-correcting algorithms of Bellman (1958) and Ford (1956) for arbitrary arc costs but no negative cycles. The primary difference between a label-correcting algorithm and a label-setting algorithm is the node selection criterion. Rather than selecting a

node with the minimum cost label to scan as the label-setting algorithm does, label-correcting algorithms may select any node from a list of eligible nodes. Using different selection criterion results in different label-correcting algorithms. Glover et al. (1985b) presented several label-correcting algorithms. Algorithms for CSP are based on these methods for SPP. Gallo and Pallottino (1986), Deo and Pang (1984), Bodin et al. (1982), and Golden and Magnanti (1977) provided reviews on SPP.

3.2. Complexity

Handler and Zang (1980) and Jaffe (1984) showed that RCSP is NP-hard, even if the graph is acyclic, only one resource constraint is involved, and all resource requirements and costs are positive (Dumitrescu and Boland (2003)). Hassin (1992) showed that SRCSP is polynomial solvable if arc costs or arc resource requirements are bounded.

Dror (1994) proved that an SPPTW with the requirement that the shortest path be elementary (no *node* is visited more than once on the path) is NP-hard in the strong sense and Garey and Johnson (1979) showed that an SPPTW that does not invoke this elementary path requirement is NP-hard in ordinary sense and can be solved in pseudo-polynomial time. The solution to RCSP is guaranteed to be elementary if arc costs are nonnegative and resource constraints have only upper bounds, or if the graph is acyclic (Beasley and Christofides (1989)). Beasley and Christofides (1989) gave the conditions for a general graph that guarantee the solution to RCSP to be elementary and presented MIP formulas for the elementary, simple (no *arc* is visited more than once on the path), and nonelementary versions of RCSP with multiple resource constraints.

3.3. Available algorithms

Since RCSPs are imbedded in a number of important practical problems, they have been studied rather extensively, as shown in Table 1, which gives taxonomy of available algorithms. Two families of exact algorithms have been proposed: one involves solving a relaxed problem using Lagrangian or linear relaxation and the other uses DP. Relaxation-based methods generally involve three steps: (1) compute lower and upper bounds for the optimal solution of RCSP by solving the relaxed problem, (2) use the results of the first step to reduce the network, and (3) close the gap between lower and upper bounds. Following this general outline, Handler and Zang (1980) solved a Lagrangian dual to optimality in a step (1) solution to SRCSP. To close the duality gap, they used the k^{th}-shortest path algorithm of Yen (1971) with modified arc cost $c_{ij} + w_{ij}u^*$ (where c_{ij} is the original arc cost, w_{ij} is the amount of resource required to traverse the arc, and u^* is the optimal value of the Lagrangian multiplier), to update lower and upper bounds of SRCSP until the lower bound was greater than or equal to the upper bound. They used Lagrangian relaxation with the goal of reducing the value of k and their method usually – but not always – achieves this goal.

Beasley and Christofides (1989) proposed a method based on Lagrangian relaxation to optimize MRCSP. They used subgradient optimization to (approximately) solve the Lagrangian dual as a first step and B&B to close the duality gap. Mehlhorn and Ziegelmann (2000) proposed a hull approach for solving the linear-relaxation of MRCSP and closed the gap between lower and upper bounds in one of three ways: by enumerating paths as in Hassin (1992), by applying a k^{th}-shortest path algorithm as in Handler and Zang (1980), or by enumerating paths in order of increasing reduced cost in combination with pruning unpromising paths. However, relaxation-based methods have not been applied when RCSP is used as a subproblem in CG.

Table 1. A taxonomy of available methods to solve RCSP and SPPRW

Reference	Problem Type	Method	c_{ij}	Graph	Features
Handler and Zang (1980)	SRCSP	LR	$c_{ij}\geq 0$	general	
Beasley and Christofides (1989)	MRCSP	LR/B&B	unrestricted	general	network reduction
Mehlhorn and Ziegelmann (2000)	MRCSP	LP	$c_{ij}\geq 0$	general	
Jaksch (1966)	SRCSP	DP	$c_{ij}>0$	general	
Aneja et al. (1983)	MRCSP	LS	$c_{ij}\geq 0$	general	network reduction
Hassin (1992)	SRCSP	DP	$c_{ij}>0$	acyclic	
Dumitrescu and Boland (2003)	RCSP(S&M)	variants of LS	$c_{ij}\geq 0$	general	network reduction
Desrochers and Soumis (1988a)	SPPTW	primal-dual	unrestricted	general	reoptimization
Desrochers and Soumis (1988b)	SPPTW	generalized permanent labeling (LS)	unrestricted	general	
Desrochers (1988)	SPPRW	primal-dual	unrestricted	general	
Feillet et al. (2004)	SPPRW	LC	unrestricted	general	elementary version
Jaumard et al. (1996)	SPPRW with resource windows at each arc	DP and a two-phase method	unrestricted	acyclic	tighten resource windows
Ioachim er al. (1998)	SPPTW with linear node cost	DP	unrestricted	acyclic	

Legend: LR - methods based on Lagrangian relaxation; LP - methods based on linear relaxation; LC - label-correcting; LS - label-setting.

A number of previous papers gave DPs for RCSP and SPPRW. Joksch (1966) proposed a DP approach for SRCSP and Hassin (1992) gave two exact pseudo-polynomial DP algorithms for SRCSP on an acyclic graph. Aneja et al. (1983) adapted the shortest-path label-setting scheme of Dijkstra (1959) to solve MRCSP. The label-setting approach is a generalization of Dijkstra's algorithm, which permanently labels nodes that are processed in order, based on resource consumptions. In addition to the label-setting approach, the label-correcting approach is a generalization of Ford-Bellman's algorithm; it treats each node more than once and attempts to update the labels of all nodes at each iteration. Desrochers and Soumis (1988b) presented a pseudo-polynomial time, generalized permanent labeling algorithm for SPPTW. Desrochers and Soumis (1988a) presented a primal-dual reoptimization approach for SPPTW and Desrochers (1988) generalized it to solve SPPRW. Dumitrescu and Boland (2003) investigated variants of the label-setting algorithm of Desrochers and Soumis (1988b) for both SRCSP and MRCSP, focusing on computational results. For SRCSP, they presented an improved version of the label-setting algorithm and an exact algorithm based on the weight-scaling method of Dumitrescu and Boland (2001), which scales resource requirements before applying the label-setting algorithm. In our terminology, a weight associated with an arc is called a resource requirement. Both algorithms integrate information obtained from preprocessing. They also presented an extension that integrates information from applying Lagrangian relaxation.

Recently, Feillet et al. (2004) studied SPPRW, restricting the optimal path to be elementary, even if negative cost cycles exist. Dror (1994) proved that the elementary SPPRW is NP-hard in the strong sense. Feillet et al. (2004) proposed an exact DP algorithm adapted from the label-correcting algorithm of Desrocher (1988), which was developed for the nonelementary path version of SPPRW, and employed it in a CG approach for VRPTW. The main advantage of the nonelementary SPPTW is that it can be solved effectively using DP, while

the primary disadvantage is that it provides a weaker lower bound. If the graph is acyclic as in this dissertation, the optimal path is always elementary.

Ioachim et al. (1998) considered a variant of SPPTW in a directed, acyclic network that includes a cost that is a linear function of the time at which service starts at each node. A certain scheduling problem, which Stojković and Soumis (2001) formulated as an integer nonlinear multicommodity network flow model with time windows and additional coupling constraints, motivated this study. Each coupling constraint links time variables (representing the service start times at nodes) in several SPPTW subproblems, each related to a single commodity. Relaxing these coupling constraints using Lagrangian relaxation (e.g., the time variables appear in the objective function together with dual multipliers) introduces a linear cost on each node in the network of each SPPTW subproblem. They proposed an exact DP algorithm for the nonelementary version of their variant of SPPTW.

In the context of CG, RCSP must be solved a number of times. Each time, arc costs are updated with the current values of dual variables in the master problem. The DP algorithms discussed above do not exploit this context. To reoptimize at each CG iteration, the DP algorithm must be employed from scratch (i.e., from the first stage and onwards). On the other hand, several approaches exploit the CG context. Jaumard et al. (1996) investigated a SPPRW in which resource windows are associated with each arc. They described a DP algorithm and a two-phase approach for an acyclic graph; both approaches have pseudo-polynomial time complexity. The first phase of their two-phase algorithm constructed an expanded graph on which SPPRW can be solved as SPP. The second phase used an (unconstrained) shortest path algorithm to prescribe an optimal path through the expanded graph. The expanded graph need be constructed only once, and SPP is solved each time a subproblem is invoked in an attempt to generate an improving column. But they did not report the numerical performance of their

method. Wilhelm et al. (2003) reported a similar, two-phase approach for a layered acyclic graph in which each arc is incident from a node in one level to another node in the next level. This dissertation develops a systematic method for solving RCSP and SPPRW repeatedly.

3.4. Preprocessing techniques

Preprocessing techniques, which aim to reduce the size of an instance either by removing nodes/arcs or by tightening resource limitations, can facilitate solution and have been used widely. They involve either resource-based reduction (Aneja et al. (1983)) or joint resource- and cost-based reduction (Beasley and Christofides (1989); Dumitrescu and Boland (2003)). Recently, Dumitrrescu and Boland (2003) presented a preprocessing algorithm for the general RCSP; it interleaves resource- and cost-based reduction, and their computational tests demonstrated that their approach can be surprisingly effective in reducing problem size. For SPPTW, Desrochers et al. (1992) reduced the time window for each node, using a resource-based method which examines and updates the time windows of nodes cyclically until no further reductions are possible. Dumas et al. (1991) reported a similar time-window tightening technique and Jaumard et al. (1996) used a forward recursion to tighten resource windows. Desaulniers et al. (2002) mentioned that tightening resource windows is a way to speed up SPPTW algorithms.

This dissertation develops an efficient preprocessing stage that implements a resource-based reduction (see Section 5.1). It does not propose a cost-based reduction because it deals with the dynamic RCSP for which arc costs are updated at each CG iteration.

3.5. Reoptimization

The objective function coefficients of a RCSP subproblem in CG are updated using the new dual variables generated by the master problem at each CG iteration. Thus, RCSP must be reoptimized at each CG iteration using the new arc costs but subject to the same set of resource constraints. The optimal solution (i.e., a shortest path tree rooted at source node) from the previous CG iteration is available.

In early work related to reoptimizing SPP, Goto and Sangiovanni-Vincentelli (1978) investigated the problem of updating shortest paths from all nodes to a set of nodes for the case in which the cost on each of a subset of arcs is decreased and proposed a method based on LU factorization of $\{c_{ij}\}$, where c_{ij} is the cost on arc (i, j). Their method requires a considerable amount of memory and is effective only if matrix $\{c_{ij}\}$ is sparse.

Gallo (1980) proposed the first efficient strategy for reoptimizing a SPP in two cases: (1) a different node is selected to be the source node of SPP each time, and (2) exactly one arc is assigned a new cost that is less than the old one each time. Fujishige (1981) proposed another effective approach for the case in which each of a set of arcs incident to a common node is assigned a new cost that is less than the previous one. Recently, Buriol et al. (2003) proposed a technique that can reduce the sizes of heaps used by several reoptimization algorithms (i.e., Ramalingam and Reps (1996), King and Thorup (2001), and Demetrescu (2001)) for the case in which a single arc is assigned a new cost that is either smaller or greater than the previous one. They provided a survey of research that had dealt with the case in which the cost of a single arc is changed and presented a comprehensive computational evaluation of their technique. These methods, however, are very restrictive and only applicable to special cases (e.g., exactly one arc is assigned a new cost, or each of a set of arcs incident to a common node is assigned a new cost). Although these methods can be applied iteratively if several arc costs change, the

resulting algorithms are computationally impractical when a number of arc costs are assigned new values.

Pallottino and Scutellà (2003) proposed a methodology to reoptimize a single-source SPP on a general network for the case in which the cost on each of any subset of arcs is changed, either to a lower or a higher value. They generalized the works of Fujishige (1981) and Gallo (1980), devising a two-phase method. The dual phase reoptimizes arcs with increased cost sequentially. The primal phase dynamically decomposes the set of arcs with decreased costs into disjoint subsets and reoptimizes each subset sequentially.

This line of research has not been specialized to the acyclic graph. This dissertation presents methods to reoptimize RCSP if the costs of any subset of arcs of the input acyclic graph are changed to values that are either higher or lower than the previous ones.

3.6. Fixed arcs

Arcs in a subproblem graph that are forbidden or prescribed correspond to associated decision variables that are fixed to either 0 or 1, respectively. This issue arises when a CG/B&B approach employs RCSP subproblem(s). Typically, existing methods assign a large cost to each forbidden arc and a small cost to each prescribed arc (e.g., Jaumard et al. (1996)) to induce the RCSP solution algorithm to exclude or include a decision variable, respectively. However, such methods have the same complexity, whether arcs are forbidden and prescribed or not, even though these constraints can be used to reduce the size of RCSP to improve computational effectiveness. This dissertation presents an algorithm that can detect infeasibility with respect to resource limitations before solving RCSP and reduce computational burden by exploiting the set of fixed arcs. Again, this method is suitable for *repeatedly* solving RCSP, subject to a set of fixed arcs at a node in the B&B search tree.

3.7. Summary

A number of applications that involve RCSP as a subproblem in CG motivate the development of methods that are explicitly designed to reoptimize in CG effectively while systematically incorporating special issues (e.g., preprocessing techniques, reoptimization, and fixed arcs). To the best of our knowledge, such methods have not been studied. This dissertation studies such a problem, which involves an acyclic underlying graph, multiple resource constraints, repeated reoptimization, and fixed arcs as required in CG/B&B applications.

CHAPTER IV

PROBLEM FORMULATION

The following notation is used in the remainder of Part I. Let $G=(V,A)$ be an acyclic digraph with $n=|V|$ topologically ordered nodes $v_1,\cdots,v_n$ and $m=|A|$ arcs. An arc from v_i to v_j is denoted as either (v_i,v_j) or $(i,j)\in A$. Since nodes of G are topologically numbered, arc $(i,j)\in A$ only if $1\le i<j\le n$. Let v_1 and v_n be source and sink nodes, respectively. An $|\Re|$-dimensional resource limit vector $\mathbf{T}=(T_1,\cdots,T_{|\Re|})$ is associated with set of nonnegative discrete valued resources, $\Re$. A cost c_{ij} and a discrete-valued resource-requirement vector $\mathbf{u}_{ij}=(u_{ij1},\cdots,u_{ij|\Re|})$ are associated with each arc $(i,j)\in A$. Traversing arc (i,j) consumes an amount u_{ijr} of resource $r\in\Re$. Let path v_1-v_j denote a series of consecutive arcs from v_1 to v_j (such a path may not be unique). The consumption (requirement) of resource r on a path is the sum of the requirements of resource r associated with all arcs on that path. Note that since the graph is assumed to be acyclic, the solution must be elementary.

RCSP consists of finding a shortest path from source node of v_1 to sink node of v_n in G with resource-limitation constraints. Let x_{ij} be a binary decision variable associated with arc $(i,j)\in A$. $x_{ij}=1$ if arc (i,j) is on the optimal shortest path, 0 otherwise. RCSP can be formulated as a MIP $(\wp 1)$:

$$(\wp 1) \qquad \min \quad z=\sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1a}$$

$$\text{s.t.} \quad \sum_{(i,j)\in A} x_{ij}-\sum_{(j,k)\in A} x_{jk}=\begin{cases}1 & j=1\\ 0 & j=2,\cdots n-1\\ -1 & j=n\end{cases} \tag{1b}$$

$$\sum_{(i,j)\in A} u_{ijr} x_{ij} \le T_r \qquad \forall\, r \in \Re \tag{1c}$$

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

Objective function (1a) is to find a shortest path with respect to arc costs, c_{ij}. Constraint (1b) requires that the solution vector $\mathbf{x} = \{ x_{ij} \}$ describes a path from v_1 to v_n. Constraint (1c) invokes the resource limitations, requiring that the total consumption of resource r on a $v_1 - v_n$ path be no greater than resource limitation T_r. Without constraint (1c), objective (1a) and constraints (1b) and (1d) define a classical SPP.

SPPRW is to find a shortest $v_1 - v_n$ path in *G* with resource-window constraints. Let t_{jr} be a decision variable representing the cumulative requirement of resource $r \in \Re$ (CRR-*r*) at $v_j \in V$ and $\mathbf{t}_j = (t_{j1}, \cdots, t_{j|\Re|})$ be a CRR vector at $v_j \in V$. Let $[\underline{t}_{jr}, \overline{t}_{jr}]$ be a hard resource window for resource $r \in \Re$ at v_j. Denote vectors $\underline{\mathbf{t}}_j = (\underline{t}_{j1}, \cdots, \underline{t}_{j|\Re|})$ and $\overline{\mathbf{t}}_j = (\overline{t}_{j1}, \cdots, \overline{t}_{j|\Re|})$. Then, SPPRW can be formulated as a MIP $(\wp 2)$:

$$(\wp 2) \qquad \min \quad z = \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{2a}$$

$$\text{s.t.} \quad \text{Constraints (1b) and (1d)}$$

$$\underline{t}_{jr} \le t_{jr} \le \overline{t}_{jr} \qquad j = 1, \ldots n\,,\ \forall\, r \in \Re \tag{2b}$$

$$t_{ir} + u_{ijr} - t_{jr} \le M(1 - x_{ij}) \qquad \forall (i,j) \in A\,,\ \forall\, r \in \Re \tag{2c}$$

where M is a large number. Objective (2a) is the same as (1a). Constraint (2b) imposes a resource window on the CRR vector at each $v_j \in V$. Constraint (2c) requires that CRR-*r* at v_j must lie within the hard resource window. If CRR-r along a path from v_1 to v_j is lower than $\underline{t}_{jr}$, it is increased to $\underline{t}_{jr}$. The difference between models $(\wp 1)$ and $(\wp 2)$ is that $(\wp 2)$ has resource-window constraints (2b) and (2c) rather than resource-limitation constraints (1c) in $(\wp 1)$. Later, it is shown that $(\wp 1)$ can be transformed to $(\wp 2)$ by formulating resource

windows $[\underline{\mathbf{t}}_j, \overline{\mathbf{t}}_j]$ for each node in G.

When these models arise as subproblems in CG approaches, the objective function coefficient c_{ij} in (1a) and (2a) is the reduced cost associated with decision variable x_{ij}, and it is a function of dual variables associated with the master problem. c_{ij} can be negative, zero, or positive. For each CG iteration, these objective function coefficients are updated using new dual variables from the master problems, while the resource constraints do not change. This induces the dynamic property of the problems studied in this dissertation.

This dissertation presents a set of solution algorithms and procedures for a set of problem types; we designate each using an acronym. For reader convenience, we give an overview of these acronyms in Table 2. The first category of Table 2 gives the acronyms for different problem types; the second category, the acronyms for algorithms and procedures; and the third category, the others.

Table 2. Main acronyms used in Part I of the dissertation

Acronyms for problem types	
KP	Knapsack problem
MCKP	Multiple-choice knapsack problem
MMCKP	Multiple-resource, multiple-choice knapsack problem
MRCSP	RCSP with multiple resource limitations
RCSP	Resource-constrained SPP (with resource limitation(s))
RC*k*SP	Resource-constrained *k*-SPP
SPP	Unconstrained shortest path problem
SPPRW	SPP with one or more types of resource windows
SPPTW	SPP with one type of resource windows (e.g., time)
SPRCRW	SPP with both resource-limitation and resource-window constraints
SRCSP	RCSP with one resource limitations
Acronyms for algorithms and procedures	
ATSA	Adaptation of TSA for solving SPPRW
EGA	Expanded-graph approach
EP	Expansion procedure used in stage 2 of TSA
GERA	G_E revising algorithm
GFA	Generating $\hat{F}_0$ algorithm

Table 2. Continued

GTSA	Generalization of TSA for solving SPRCRW
LSA	Label-setting algorithm for solving RCSP
MDFA	Method for dealing with fixed arcs
OA	Optimizing algorithm used in stage 3 of TSA
ROA	Reoptimizing algorithm used in stage 3 of TSA
RP	Rounding procedure used in EP
S1A	Stage 1 algorithm used in TSA
S1A-A	Stage 1 algorithm used in ATSA
S1A-G	Stage 1 algorithm used in GTSA
S1A-M	Stage 1 algorithm used in TSA for an MMCKP-graph
TSA	Three-stage approach
TSA-CG	A version of TSA for solving RCSP in CG
TSA-CG/B&B	A version of TSA for solving RCSP in CG/B&B
Other acronyms	
CRR-*r*	Cumulative requirement of resource *r*
CRR vector	Cumulative resource requirement vector

This dissertation proposes a three-stage approach (TSA), which comprises a preliminary phase (stages 1 and 2) and an iterative solution phase. We may select algorithms for each stage for different problem types. For reader convenience, Table 3 gives an overview of the algorithms we present for each stage of TSA to solve different problem types. Later chapters discuss each.

Table 3. TSA for different problem types

	RCSP SRCSP MRCSP	SPPRW SPPTW	SPRCRW	RC*k*SP	MMCKP MCKP KP
Preliminary phase					
Stage 1: (preprocessing)	S1A	S1A-A	S1A-G	S1A	S1A or S1A-M
Stage 2: (expanding)	EP				
(dealing with fixed arcs)	MDFA or traditional method				
Iterative solution phase					
Stage 3: (solving)	OA or ROA			Unconstrained *k*-SPP algorithm	OA or ROA

CHAPTER V

THREE-STAGE APPROACH

This chapter introduces a new, improved approach (TSA) for solving RCSP and analyses its complexity. TSA comprises three stages and is especially suitable for iterative reoptimization in the context of CG. Section 5.1 describes the first stage – preprocessing – in which a network reduction technique is used repeatedly to delete the nodes and arcs that can not be on any feasible path. Importantly, tight resource windows are prescribed for each node in the reduced graph via the preprocessing stage so that RCSP becomes SPPRW on the reduced graph. Section 5.2 presents the second stage – an expanding stage, in which SPPRW on the reduced graph is transformed into SPP via a special procedure. Since stages 1 and 2 are one-time processes; we include them both in what we call the preliminary phase. Section 5.3 presents the third stage – an iterative solution stage that incorporates an algorithm for finding a shortest path with respect to a given set of arc costs. Following that, Section 5.4 presents a combination of algorithms that form a TSA for repeatedly solving RCSP in the context of CG. Figure 1 shows a general outline of TSA. Finally, Section 5.5 gives a summary of this chapter.

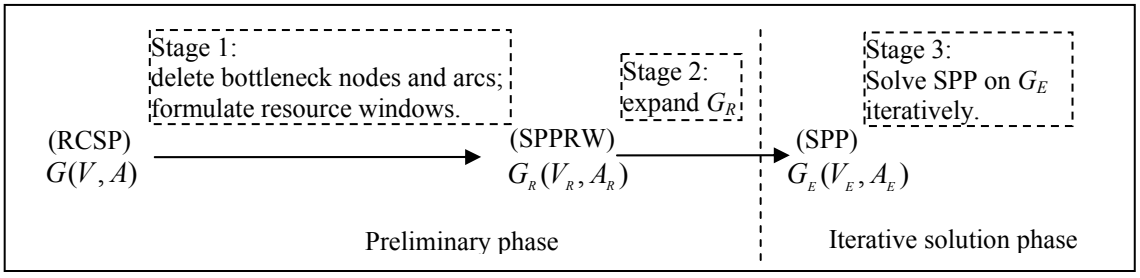


Fig. 1. General outline of TSA.

5.1. Preprocessing stage

Stage 1 (preprocessing) uses a resource-based network reduction technique to delete nodes and arcs that can not be on any feasible (i.e., with respect to the resource limitations) path. This method does not use any information about arc costs and is, thus, valid for any set of arc costs. This study does not explore a cost-based reduction because it addresses the dynamic RCSP for which arc costs are updated at each CG iteration. Further, stage 1 formulates resource windows for each node in the reduced graph, based on the resource-limitation constraints. In this section, Subsection 5.1.1 presents the stage 1 algorithm (S1A); following that, Subsection 5.1.2 shows some properties inferred from S1A; Subsection 5.1.3 analyzes the complexity of S1A; and Subsection 5.1.4 provides a brief summary of stage 1.

5.1.1. Description of S1A

Before presenting S1A, we define a bottleneck arc (node) as one that can not be on a feasible $v_1 - v_n$ path. A bottleneck arc (i, j) is an arc that renders the resource available at v_j insufficient to continue a path to the sink node. A bottleneck node has no arcs incident to it (both incoming and outgoing arcs) that can be on a feasible path to the sink node. Thus, all arcs incident to a bottleneck node are bottleneck arcs. Denote $G_R = (V_R, A_R)$ as the reduced graph obtained from stage 1 by deleting bottleneck nodes and arcs from G. Let $FS(v_j)$ and $BS(v_j)$ denote the sets of the successors (forward stars) and predecessors (backward stars) of v_j in G_R, respectively, that is, $FS(v_j) = \{v_i : (v_j, v_i) \in A_R\}$ and $BS(v_j) = \{v_i : (v_i, v_j) \in A_R\}$.

We now introduce additional notation that we use to present S1A. As Figure 2 depicts, some $v_1 - v_j$ path requires the least (most) amount of resource $r \in \Re$ in comparison with all other $v_1 - v_j$ paths. Let $\underline{f}_{jr}$ ($\overline{f}_{jr}$) denote this minimum (maximum) resource requirement.

Similarly, some $v_j - v_n$ path requires the least (most) amount of resource r over all $v_j - v_n$ paths. Let $\underline{a}_{jr}$ ($\overline{a}_{jr}$) denote the corresponding minimum (maximum) amount of resource r required for traversing from v_j to v_n and $\underline{b}_{jr} = T_r - \overline{a}_{jr}$ ($\overline{b}_{jr} = T_r - \underline{a}_{jr}$) denote the corresponding minimum (maximum) amount of resource r that would be left over (i.e., available) for a $v_1 - v_j$ path.

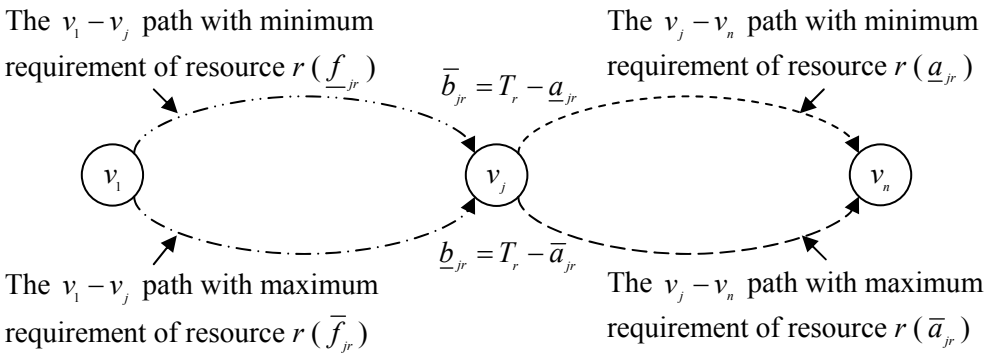


Fig. 2. Illustration of notation $\underline{f}_{jr}$, $\overline{f}_{jr}$, $\underline{a}_{jr}$, $\overline{a}_{jr}$, $\underline{b}_{jr}$ and $\overline{b}_{jr}$.

Figure 3 details S1A, in which steps 1-4 detail the network reduction technique, which identifies and deletes bottleneck nodes and arcs and step 5 formulates tight resource windows for each node in G_R. Steps 3 - 4 identify and delete bottleneck nodes and arcs iteratively until no more bottleneck nodes or arcs can be deleted. The backward pass determines $\overline{b}_{jr}$, the maximum amount of resource r left over (i.e., available) for all $v_1 - v_j$ paths (for node v_j) relative to resource limitation T_r. The forward pass determines $\underline{f}_{jr}$, the minimum amount of resource r required over all $v_1 - v_j$ paths. In backward pass, node v_j with $\overline{b}_{jr} < 0$ (step 3(i)) is a bottleneck because the maximum amount of resource r that is available for $v_1 - v_j$ paths ($\overline{b}_{jr}$) is less than 0; in addition, arc (i, j) with $u_{ijr} > \overline{b}_{jr}$ (step 3(ii)) is a bottleneck because the maximum amount

of resource r that is available for $v_1 - v_j$ paths is less than the resource requirement on arc (i,j), so arc (i,j) can not be used. In forward pass, node v_j with $\overline{b}_{jr} < \underline{f}_{jr}$ (step 3(iii)) and arc (i,j) with $\underline{f}_{ir} + u_{ijr} > \overline{b}_{jr}$ (step 3(iv)) are bottlenecks for similar reasons. The conditions for identifying bottleneck nodes and arcs in forward pass are tighter than the ones in backward pass because $\underline{f}_{jr}$ are calculated in the forward pass.

(Network reduction)

step 1. Topologically sort the nodes of G: $v_1, \cdots, v_n$.

step 2. Initialize $G_R = G$ by setting $V_R = V$ and $A_R = A$.

step 3. For $r = 1 \cdots |\Re|$,

set $\overline{b}_{jr} = 0$, $\forall j = 1, \ldots, n-1$, $\overline{b}_{nr} = T_r$ and $\underline{f}_{1r} = 0$, $\underline{f}_{jr} = T_r$, $\forall j = 2, \ldots, n$.

(backward pass) For each $v_j \in V_R \setminus \{v_n\}$ in decreasing v_j index,

calculate $\overline{b}_{jr} = \max_{i \in FS(v_j)} \left\{ \overline{b}_{ir} - u_{jir} \right\}$ (if $FS(v_j) = \varnothing$, $\overline{b}_{jr} = -T_r$);

(i) if $\overline{b}_{jr} < 0$, v_j is bottleneck; delete v_j and all the arcs incident to it;

(ii) if $u_{jir} > \overline{b}_{ir}$, arc (v_j, v_i) is a bottleneck; delete it.

If v_1 is bottleneck, STOP; RCSP is infeasible.

(forward pass) For each $v_j \in V_R \setminus \{v_1\}$ in increasing v_j index,

calculate $\underline{f}_{jr} = \min_{i \in BS(v_j)} \left\{ \underline{f}_{ir} + u_{ijr} \right\}$ (if $BS(v_j) = \varnothing$, $\underline{f}_{jr} = T_r$);

(iii) if $\overline{b}_{jr} < \underline{f}_{jr}$, v_j is bottleneck; delete v_j and all the arcs incident to it;

(iv) if $\underline{f}_{ir} + u_{ijr} > \overline{b}_{jr}$, arc (v_i, v_j) is a bottleneck; delete it.

If v_n is bottleneck, STOP; RCSP is infeasible.

step 4. If anything was deleted from the graph, go back to step 3.

(Formulate resource windows)

step 5. For $r = 1 \cdots |\Re|$,

set $\underline{b}_{nr} = T_r$, $\overline{f}_{1r} = 0$.

For each $v_j \in V_R \setminus \{v_n\}$ in decreasing v_j index, calculate $\underline{b}_{jr} = \min_{i \in FS(v_j)} \left\{ \underline{b}_{ir} - u_{jir} \right\}$.

For each $v_j \in V_R \setminus \{v_1\}$ in increasing v_j index, calculate $\overline{f}_{jr} = \max_{i \in BS(v_j)} \left\{ \overline{f}_{ir} + u_{ijr} \right\}$;

(i) if $\overline{f}_{jr} > \underline{b}_{jr}$, $\underline{t}_{jr} = \max\left\{ \underline{f}_{jr}, \underline{b}_{jr} \right\}$ and $\overline{t}_{jr} = \min\left\{ \overline{f}_{jr}, \overline{b}_{jr} \right\}$, else $\underline{t}_{jr} = \overline{t}_{jr} = \overline{f}_{jr}$.

STOP.

Fig. 3. S1A.

Step 5 establishes the window $[\underline{t}_{jr}, \overline{t}_{jr}]$ for each resource $r \in \Re$ at each node v_j that remains in G after deleting bottleneck nodes and arcs, that is $V(G_R) = V_R$. Recall that $\underline{t}_{jr}$ and $\overline{t}_{jr}$ are the lower and upper bounds, respectively, for CRR-*r* at v_j as defined in model $(\wp 2)$. The window $[\underline{t}_{jr}, \overline{t}_{jr}]$ is determined as follows. In step 5(i), if $\overline{f}_{jr} \le \underline{b}_{jr}$, then every $v_1 - v_n$ path containing v_j is feasible and so $\underline{t}_{jr} = \overline{t}_{jr} = \overline{f}_{jr}$. If $\overline{f}_{jr} > \underline{b}_{jr}$, $\overline{t}_{jr} = \min\{\overline{b}_{jr}, \overline{f}_{jr}\}$ and $\underline{t}_{jr} = \max\{\underline{b}_{jr}, \underline{f}_{jr}\}$. The resource windows established in this way are hard resource windows. That is, any $v_1 - v_j$ path with CRR-*r* larger than $\overline{t}_{jr}$ is infeasible; and if CRR-*r* along a $v_1 - v_j$ path is lower than $\underline{t}_{jr}$, then CRR-*r* at v_j is increased to $\underline{t}_{jr}$.

We interpret the calculations of $\overline{t}_{jr}$ and $\underline{t}_{jr}$ for the case in which $\overline{f}_{jr} > \underline{b}_{jr}$ through a numerical example. Considering $\overline{t}_{jr}$, if $\overline{f}_{jr} = 10$ and $\overline{b}_{jr} = 12$, then $\overline{t}_{jr} = \min\{10,12\} = 10 = \overline{f}_{jr}$ and CRR-*r* along any $v_1 - v_j$ path is less than or equal to $\overline{t}_{jr}$. If $\overline{f}_{jr} = 10$ and $\overline{b}_{jr} = 8$, then $\overline{t}_{jr} = \min\{10,8\} = 8 = \overline{b}_{jr}$ and any $v_1 - v_j$ path with CRR-*r* larger than 8 would not be augmented to a feasible $v_1 - v_n$ path because $\overline{b}_{jr} = 8$. For $\underline{t}_{jr}$, if $\underline{b}_{jr} = 5$ and $\underline{f}_{jr} = 7$, then $\underline{t}_{jr} = \max\{5,7\} = 7 = \underline{f}_{jr}$ and CRR-*r* along any $v_1 - v_j$ path is no less than $\underline{t}_{jr} = 7$. If $\underline{b}_{jr} = 5$ and $\underline{f}_{jr} = 3$, then $\underline{t}_{jr} = \max\{5,3\} = 5 = \underline{b}_{jr}$ and if CRR-*r* on a $v_1 - v_j$ path is less than 5, it is increased to 5 because $\underline{b}_{jr} = 5$, i.e., 5 is enough to traverse from v_j to v_n on any $v_j - v_n$ path. Note that, $\overline{\mathbf{t}}_1 = \underline{\mathbf{t}}_1 = \mathbf{0}$ and $\overline{\mathbf{t}}_n = \underline{\mathbf{t}}_n = \mathbf{T}$. Proposition 5.4, which is proven in the next subsection, establishes the correctness of step 5.

5.1.2. Properties

Proposition 5.1 can be used to judge whether RCSP is infeasible or not and Proposition 5.2 can be used to eliminate some resource constraint(s) from the problem.

Proposition 5.1. The following statements are equivalent: (a) v_1 is bottleneck; (b) v_n is bottleneck; (c) every node of *G* is bottleneck; and (d) RCSP is resource infeasible (i.e., no $v_1 - v_n$ path satisfies all resource constraints).

Proof. Straightforward by the definition of bottleneck node. ■

Proposition 5.2. If $\underline{b}_{1r} \geq 0$ or $\overline{f}_{nr} \leq T_r$, then the limitation of resource $r \in \Re$ is redundant. If $\underline{b}_{1r} \geq 0$ or $\overline{f}_{nr} \leq T_r$ for all $r \in \Re$, then all $v_1 - v_n$ paths in G_R are feasible with respect to the resource constraints and RCSP reduces to SPP.

Proof. Recall that $\overline{a}_{1r}$ is the maximum requirement of resource r over all $v_1 - v_n$ paths. Thus, $\overline{a}_{1r} = \overline{f}_{nr}$. Because $\underline{b}_{1r} = T_r - \overline{a}_{1r}$, $\underline{b}_{1r} \geq 0$ implies $\overline{a}_{1r} \leq T_r$ and, equivalently, $\overline{f}_{nr} \leq T_r$. $\overline{f}_{nr} \leq T_r$ implies every $v_1 - v_n$ path is feasible with respect to resource r, thus, the limitation of resource $r \in \Re$ is redundant and can be eliminated. If all resource constraints are redundant, then RCSP reduces to SPP. ■

The next proposition is valid for SRCSP (i.e., $|\Re| = 1$).

Proposition 5.3. For SRCSP, graph G need be traversed only once in the backward direction and only once in the forward direction to delete all bottleneck nodes and arcs.

Proof. We omit subscript r in this proof because SRCSP entails only one type of resource. Note that if a node is bottleneck, all the arcs incident to it (both incoming and outgoing arcs) are bottleneck arcs. Thus, deleting a bottleneck node is equivalent to deleting all arcs incident to it. Thus, we only need to show that the claim is true for bottleneck arcs.

First, we show that for SRCSP, an arc (i, j) is bottleneck if and only if $\underline{f}_i + u_{ij} > \overline{b}_j$. To see that, if $\underline{f}_i + u_{ij} > \overline{b}_j$, it follows that $\underline{f}_i + u_{ij} + \underline{a}_i > T_r$ and (i, j) is bottleneck arc. On the other hand, if arc (i, j) is a bottleneck, then $\underline{f}_i + u_{ij} + \underline{a}_i > T_r$, i.e., $\underline{f}_i + u_{ij} > \overline{b}_j$. By this observation, to prove the proposition, it is sufficient to show that the deletion of bottleneck arcs will not change the $\underline{f}_j$ and $\overline{b}_j$ values that have already been established upon the completion of a backward and a forward pass. The calculation of $\overline{b}_j$ involves only nodes $v_j, \ldots, v_n$, so no bottleneck arc deleted during the backward pass will change the established $\overline{b}_j$ values; similarly, no bottleneck arc deleted during the forward pass will change the established $\underline{f}_j$ values.

Now, we show that no bottleneck arc deleted during the forward pass will change the established $\overline{b}_j$ values. Suppose that bottleneck arc (i, j) is identified and deleted during the forward pass. Since the calculation of $\overline{b}_p$ involves only nodes $v_p, \ldots, v_n$, $\overline{b}_p$ does not change for any $p = i+1, \ldots n$. Now, consider $\overline{b}_p$ for v_p ($p = 1, \ldots i$) in the current (perhaps partially) reduced graph.

By way of contradiction, suppose $\overline{b}_p$ is changed because arc (i, j) is deleted. Then, arc (i, j) must be on the $v_p - v_n$ path with minimum resource consumption and, consequently, $\overline{b}_j - u_{ij} = \overline{b}_i$. Note that v_i is not a bottleneck (otherwise v_p is a bottleneck and would have been eliminated to attain the current reduced graph). Since arc (i, j) is a bottleneck arc, $\underline{f}_i + u_{ij} > \overline{b}_j$. It follows that $\underline{f}_i > \overline{b}_i$, so v_i is a bottleneck. But since v_i is not a bottleneck, $\overline{b}_p$ for $p = 1, \ldots i$ can not change, which contradicts the supposition. This completes the proof. ■

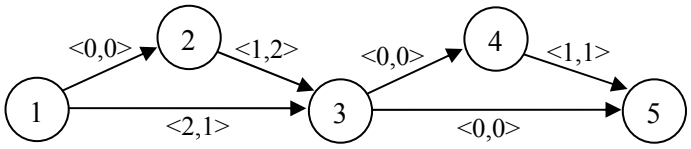
It is worth mentioning that step 3 of TSA traverses the graph first in the backward direction then in the forward direction and Proposition 5.3 is only valid for that order of

traversals. Recall that the backward pass determines the maximum amount of each resource left over (i.e., available) for all $v_1 - v_j$ paths (for each node v_j) relative to the resource limitation $\mathbf{T}$. The forward pass determines the minimum amount of each resource required over all $v_1 - v_j$ paths. During the forward pass, some bottleneck nodes and/or arcs may be deleted if the maximum resource requirement from the backward pass is less than the minimum resource requirement from the forward pass. If the forward pass were performed first, the minimum resource requirement would be the smallest possible for each node because none of the original nodes and arcs would have been deleted from the graph, regardless of the value of $\mathbf{T}$. If this initial forward pass were then followed by a backward pass that deletes any node or arc, the minimum resource requirement determined by the forward pass must be updated at each node between the node where the bottleneck arc ends and the sink node.

Although, for SRCSP, we only need to traverse the graph twice to delete all bottleneck nodes and arcs, for MRCSP, we may traverse the graph more than twice for each resource type. To see that, suppose step 3 of S1A completes backward and forward passes for resource $r \in \Re$ first and for resource $r' \in \Re$ second, where $r' \neq r$. Further, suppose some bottleneck nodes and/or arcs are identified and deleted due to resource limitation $T_{r'}$ to revise the reduced graph. Then, values of $\underline{f}_{jr}$ and $\overline{b}_{jr}$ may change when they are recalculated relative to this revised reduced graph. Consequently, it may be possible to identify additional bottleneck nodes and arcs due to resource limitation T_r. Thus, for MRCSP, step 3 should be repeated until no more bottleneck nodes and arcs can be deleted.

For SRCSP, Proposition 5.3 guarantees that S1A can identify and delete *all* bottleneck nodes and arcs. However, for MRCSP, it can not guarantee that it will identify all bottlenecks. Figure 4 depicts an example of MRCSP in which S1A does not identify all bottleneck nodes and

arcs. This example comprises 5 nodes and 6 arcs with 2 types of resources and $T_1 = T_2 = 2$. Here, label <a,b> on an arc gives the requirements for resources 1 and 2, respectively. The table on the right side in Figure 4 gives the values of $\overline{b}_{jr}$ and $\underline{f}_{jr}$ that are calculated by step 3 in S1A. Arcs (3,4) and (4,5) and node 4 are bottlenecks, but S1A can not identify them as such. The reason is that S1A identifies bottleneck nodes and arcs based on the values of $\underline{f}_{jr}$ and $\overline{b}_{jr}$ for each *individual* resource. But this is not sufficient for MRCSP. As shown in the example, neither arc (3,4) nor arc (4,5) can be identified as a bottleneck because $\underline{f}_{3,r} + u_{3,4,r} = 1 + 0 = \overline{b}_{4,r} = 1$ and $\underline{f}_{4,r} + u_{4,5,r} = 1 + 1 = \overline{b}_{5,r} = 2$, for $r = 1, 2$. But, in fact, either of the two possible CRR vectors at node 3 – $< 1, 2 >$ or <2,1> – together with arcs (3,4) and (4,5), result in infeasible $v_1 - v_5$ paths with respect to resource limitations.



First cycle of step 3 in S1A					
v_j	v_1	v_2	v_3	v_4	v_5
$\overline{b}_{j1}$	1	1	2	1	2
$\underline{f}_{j1}$	0	0	1	1	1
$\overline{b}_{j2}$	1	0	2	1	2
$\underline{f}_{j2}$	0	0	1	1	1
No bottleneck nodes and arcs are identified and deleted, STOP.					

Fig. 4. An example of MRCSP.

Proposition 5.4. For each v_j in G_R, (a) any $v_1 - v_j$ path with CRR-*r*, for $r \in \Re$, larger than $\overline{t}_{jr}$ can not be on a feasible $v_1 - v_n$ path, and (b) if CRR-*r* associated with a $v_1 - v_j$ path is less than $\underline{t}_{jr}$, it can be increased to $\underline{t}_{jr}$.

Proof. For part (a), if $\overline{f}_{jr} \le \underline{b}_{jr}$, then $\overline{t}_{jr} = \overline{f}_{jr}$ and no $v_1 - v_j$ path has a larger CRR-*r* than $\overline{t}_{jr}$; we are done. Otherwise, $\overline{t}_{jr} = \min\left\{\overline{b}_{jr}, \overline{f}_{jr}\right\}$ and there are two cases. Case (a1) is that, if $\overline{b}_{jr} \ge \overline{f}_{jr}$, then $\overline{t}_{jr} = \overline{f}_{jr}$ and no $v_1 - v_j$ path has a larger CRR-*r* than $\overline{t}_{jr}$; we are done. Case (a2) is that, if $\overline{b}_{jr} < \overline{f}_{jr}$, then $\overline{t}_{jr} = \overline{b}_{jr}$. Suppose there is a $v_1 - v_j$ path P with a CRR-*r* of μ_{jr} with $\mu_{jr} > \overline{t}_{jr} = \overline{b}_{jr}$. Recall that $\overline{b}_{jr} = T_r - \underline{a}_{jr}$ where $\underline{a}_{jr}$ is the minimum requirement of resource r over all $v_j - v_n$ paths. Then $\mu_{jr} + \underline{a}_{jr} > T_r$ (i.e., P can not be a part of any feasible $v_1 - v_n$ path).

For part (b), if $\overline{f}_{jr} > \underline{b}_{jr}$, then $\underline{t}_{jr} = \max\left\{\underline{b}_{jr}, \underline{f}_{jr}\right\}$ and there are two cases. Case (b1) is that, if $\underline{b}_{jr} < \underline{f}_{jr}$, then $\underline{t}_{jr} = \underline{f}_{jr}$ and no $v_1 - v_j$ path has CRR-*r* less than $\underline{t}_{jr}$; we are done. Case (b2) is that, if $\underline{b}_{jr} > \underline{f}_{jr}$, then $\underline{t}_{jr} = \underline{b}_{jr}$. Suppose there is a $v_1 - v_j$ path P with CRR-*r* of μ_{jr} with $\mu_{jr} < \underline{t}_{jr} = \underline{b}_{jr}$. Recall that $\underline{b}_{jr} = T_r - \overline{a}_{jr}$ where $\overline{a}_{jr}$ is the maximum requirement of resource r over all $v_j - v_n$ paths. Then $\mu_{jr} + \overline{a}_{jr} < T_r$ (i.e., P augmented by any $v_j - v_n$ path results in a feasible $v_1 - v_n$ path). By setting μ_{jr} to $\mu'_{jr} = \underline{t}_{jr} = \underline{b}_{jr}$, it follows that $\mu'_{jr} + \overline{a}_{jr} = T_r$; that is, P can still be augmented by any $v_j - v_n$ path to form a feasible $v_1 - v_n$ path. If $\overline{f}_{jr} \le \underline{b}_{jr}$, $\underline{t}_{jr} = \overline{f}_{jr}$ and the statement can be proven true by applying an illustration that is similar to that used in case (b2). This completes the proof. ■

Corollary 5.1. For SRCSP, each node and arc in G_R must be on some $v_1 - v_n$ path(s) that satisfy resource windows determined by S1A.

Proof. By Propositions 5.3 and 5.4. ■

By Proposition 5.4, the original RCSP on G is reduced to SPPRW on G_R with hard resource windows $[\underline{t}_{jr}, \overline{t}_{jr}]$, for $v_j \in V_R$ and $r \in \Re$. Then, the optimal value of RCSP on G is

equal to the optimal value of SPPRW on G_R. SPPRW, as depicted by model $(\wp 2)$ (Chapter IV), is to find a shortest path from v_1 to v_n on acyclic digraph $G_R(V_R, A_R)$ that satisfies the resource-window constraints.

5.1.3. Computational complexity of S1A

Let γ be the number of times (i.e., cycles) step 3 is repeated.

Proposition 5.5. S1A runs in $O(|\Re| m\gamma)$ time, where $m = |A|$ is the number of arcs in G. For SRCSP, S1A runs in $O(|\Re| m)$.

Proof. Assume $|A| > |V| = n$. Step 1, topological sorting of nodes, can be done in $O(m)$ time; step 3 runs in $O(|\Re| m)$ time for each cycle because every arc in G is processed in constant time for each resource $r \in \Re$; step 5 runs in $O(|\Re| m)$ time for the same reason. Thus, the total time for S1A is $O(|\Re| m\gamma)$. For SRCSP, by Proposition 5.3, we have $\gamma = 1$; thus, the run time of S1A is $O(|\Re| m)$. ■

For SRCSP, $\gamma = 1$ by Proposition 5.3. Theoretically, for MRCSP, $|A|$ is an upper bound for γ. Because, in the worst case, each cycle of step 3 identifies and deletes only one bottleneck arc and the maximum number of bottleneck arcs is $|A|$. However, on average, γ is much smaller than $|A|$. Based on the computational results that Chapter VII presents, 1-6 cycles of step 3 are sufficient for preprocessing MRCSP.

5.1.4. Summary

Stage 1 (i.e., S1A) has two functions. For SRCSP, it deletes all bottleneck nodes and arcs and, for MRCSP, it deletes some, if not all, of the bottleneck nodes and arcs, using a resource-based network reduction technique. For MRCSP, step 3 is repeated until no more bottleneck nodes or

arcs can be deleted. In contrast, for SRCSP, Proposition 5.3 shows that steps 3-4 need be implemented only once to identify and delete all bottleneck nodes and arcs. This network reduction technique can be included in a preprocessing stage to enhance any algorithm for solving RCSP on an acyclic graph to help reduce the computational burden.

Second, S1A tightens resource windows at each node in the reduced graph G_R. After stage 1, RCSP on input graph G becomes SPPRW on reduced graph G_R. In RCSP, a difficulty arises because it is not possible to determine if constraints (1c) are satisfied or violated until an entire path from the source node to the sink node has been defined. Thus, the infeasibility of a path can only be determined very late in the construction of the path. Changing resource-limitation constraints (1c) to resource-window constraints (2b) and (2c) by specifying a window for each resource at each node overcomes this difficulty.

5.2. Expanding stage

Stage 1 of S1A reduces the original RCSP on G to SPPRW on G_R with hard resource windows $[\underline{\mathbf{t}}_j, \overline{\mathbf{t}}_j]$ for $v_j \in V_R$. Starting from SPPRW on G_R, stage 2 uses an expansion procedure (EP) to transform SPPRW on G_R to SPP on expanded graph $G_E = (V_E, A_E)$. That is, RCSP on G is reduced to SPP on G_E by stage 2. This section proposes EP for stage 2 and introduces additional notation in Subsection 5.2.1, presents some properties inferred from EP in Subsection 5.2.2, analyzes the complexity of EP in Subsection 5.2.3, and gives a brief summary of this section in Subsection 5.2.4.

5.2.1. Description of EP

EP applies to $G_R(V_R, A_R)$ with resource windows $[\underline{\mathbf{t}}_j, \overline{\mathbf{t}}_j]$, for $v_j \in V_R$. Since V_R is obtained by deleting bottleneck nodes from V, the indices of nodes in V_R are still in topological order (i.e., arc $(v_i, v_j) \in V_R$ if and only if $i < j$) but they may not be consecutive.

Now, we introduce additional notation used to detail EP. Let S_j be a set of nodes in expanded graph G_E that is associated with specific node $v_j \in V_R$ in G_R. Let $s_j^k \in S_j \subseteq V_E$ denote the k^{th} node in set S_j, when $k = 1, 2, \ldots |S_j|$. As it expands G_R to form G_E, EP may define a set of nodes S_j associated with $v_j \in V_R$ in G_R. Then, $V_E = \bigcup_{v_j \in V_R} S_j$ and $S_j \cap S_{j'} = \varnothing$, $v_j,\ v_{j'} \in V_R,\ j \neq j'$. G_E thus comprises $|V_R|$ such sets of nodes. Let $RL_{jr} = \left\{d_{jr}^1, \ldots, d_{jr}^{|RL_{jr}|}\right\}$ be an ordered set, where $d_{jr}^1 = \underline{t}_{jr}$, $d_{jr}^{|RL_{jr}|} = \overline{t}_{jr}$ and $d_{jr}^{\ell-1} < d_{jr}^{\ell}$, for $\ell = 2, \ldots, |RL_{jr}|$. For each $v_j \in V_R$ in decreasing order, RL_{jr} is calculated according to:

$$\begin{aligned} &RL_{nr} = \{T_r\}, \forall r \in \Re, \\ &RL_{jr} = \left\{ d_{jr} \mid d_{jr} = d_{ir} - u_{jir}, \underline{t}_{jr} \le d_{jr} \le \overline{t}_{jr}, d_{ir} \in RL_{ir}, v_i \in FS(v_j) \right\} \cup \left\{ \underline{t}_{jr}, \overline{t}_{jr} \right\}, \forall r \in \Re. \end{aligned} \tag{3}$$

By way of (3), each element in RL_{jr} represents an amount of resource $r \in \Re$ left over (i.e., available) for $v_1 - v_j$ paths and is restricted to lie within window $[\underline{t}_{jr}, \overline{t}_{jr}]$. Expression (3) starts from the sink node v_n and initializes $RL_{nr} = \{T_r\}$ (i.e., the amount of resource r left over for $v_1 - v_n$ paths is T_r). In decreasing order of $v_j \in V_R$, expression (3) calculates RL_{jr} until finishing RL_{1r} for the source node.

Let $\mathbf{y}_j^k = (y_{j1}^k, \cdots y_{j|\Re|}^k)$ be a CRR vector at node $s_j^k \in S_j$ and Y_j be the set of CRR vectors $\mathbf{y}_j^k$ for $s_j^k \in S_j$ such that $\underline{\mathbf{t}}_j \le \mathbf{y}_j^k \le \overline{\mathbf{t}}_j$. Each node $s_j^k \in S_j \subseteq V_E$ is associated with a unique

$\mathbf{y}_j^k \in Y_j$. Initialize $Y_1 = \{\mathbf{y}_1^1 = \mathbf{0}\}$. For each $v_j \in V_R \setminus \{v_1\}$ in increasing order, the calculation of y_{jr}^k involves two steps. First compute

$$g_{jr}^k = y_{ir}^h + u_{ijr}, \text{ for } \mathbf{y}_i^h \in Y_i,\ (i,j) \in A_R \text{ and } r \in \Re. \tag{4}$$

Then, the value of g_{jr}^k is rounded up using RL_{jr} to define y_{jr}^k according to (5):

$$y_{jr}^k = \begin{cases} d_{jr}^1 & \text{if} \quad g_{jr}^k \le d_{jr}^1 = \underline{t}_{jr}; & \text{(5a)} \\ d_{jr}^\ell & \text{if} \quad d_{jr}^{\ell-1} < g_{jr}^k \le d_{jr}^\ell \ \text{ for some } \ell = 2,\ldots,\left|RL_{jr}\right|; & \text{(5b)} \\ +\infty & \text{if} \quad g_{jr}^k > d_{jr}^{|RL_{jr}|} = \overline{t}_{jr}. & \text{(5c)} \end{cases}$$

Expression (5a) rounds g_{jr}^k up to $y_{jr}^k = d_{jr}^1 = \underline{t}_{jr}$ when g_{jr}^k is lower than $d_{jr}^1 = \underline{t}_{jr}$ (see Proposition 5.4). Expression (5b) rounds all values of g_{jr}^k on the interval $(d_{jr}^{\ell-1}, d_{jr}^\ell]$ up to $y_{jr}^k = d_{jr}^\ell$, for $\ell = 2,\ldots,\left|RL_{jr}\right|$. Figure 5 illustrates that, if a value of g_{jr}^k corresponding to a $v_1 - v_j$ path falls in interval $(d_{jr}^{\ell-1}, d_{jr}^\ell]$, this $v_1 - v_j$ path can be augmented with the same set of $v_j - v_n$ paths as CRR-r of d_{jr}^ℓ to form feasible $v_1 - v_n$ paths. Expression (5c) sets $y_{jr}^k = +\infty$ if $g_{jr}^k > d_{jr}^{|RL_{jr}|} = \overline{t}_{jr}$ because g_{jr}^k violates the limitation of resource r (Proposition 5.4) and step 3(iii) of EP (see Figure 6) will discard it.

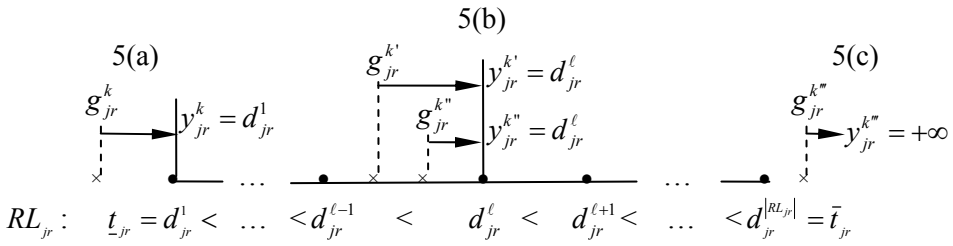


Fig. 5. y_{jr}^k is calculated using RP (4-5).

Remark 1. Let Λ_{jr} be a set of values of y_{jr}^k for $v_j \in V_R$ and $r \in \Re$. If some interval (e.g., $(d_{jr}^{\ell}, d_{jr}^{\ell+1}]$ in Figure 5), does not have any g_{jr}^k falling in it, then $d_{jr}^{\ell+1}$ will not be in Λ_{jr}. Thus, $\Lambda_{jr} \subseteq RL_{jr} \subseteq \{\underline{t}_{jr}, \cdots, \overline{t}_{jr}\}$ and $|\Lambda_{jr}| \le |RL_{jr}| \le \overline{t}_{jr} - \underline{t}_{jr} + 1$. That is, using RL_{jr} to round g_{jr}^k up to y_{jr}^k reduces the solution space, perhaps significantly. Hereafter, we refer to expressions (4) and (5) as the rounding procedure (RP).

With this background, Figure 6 details EP. EP processes nodes $v_j \in V_R$ in order of increasing index, calculating $\mathbf{y}_j^k$, associating a node $s_j^k \in S_j$ with each unique vector $\mathbf{y}_j^k = (y_{j1}^k, \cdots y_{j|\Re|}^k) \in Y_j$, and connecting s_i^h to s_j^k if $\mathbf{y}_j^k$ is calculated from $\mathbf{y}_i^h$ (in steps 3(iii-iv)).

step 1.	For each $r \in \Re$ and $v_j \in V_R$, calculate RL_{jr} according to (3).
step 2.	Initialize $S_1 = \{s_1^1\}$, $Y_1 = \{\mathbf{y}_1^1 = \mathbf{0}\}$.
step 3.	For each $v_j \in V_R \setminus \{v_1\}$ in increasing v_j index,
	$k = 1$.
	For each $v_i \in BS(v_j)$,
(i)	for each $\mathbf{y}_i^h \in Y_i$,
(ii)	for each $r \in \Re$, calculate y_{jr}^k according to RP;
(iii)	if $y_{jr}^k > \overline{t}_{jr}$, discard $\mathbf{y}_j^k$ and go to step 3(i).
(iv)	If $\mathbf{y}_j^k = \mathbf{y}_j^{k'}$ for $\mathbf{y}_j^{k'} \in Y_j$, add arc $(s_i^{h'}, s_j^k) \to A_E$;
(v)	else add $\mathbf{y}_j^k \to Y_j$ in lexicographic order; $s_j^k \to S_j$; $(s_i^h, s_j^k) \to A_E$; $k \leftarrow k+1$.
(vi)	Set $V_E = V_E \cup S_j$ and free the memory used to store RL_{jr}.
step 4.	If $\|\Re\| > 1$, delete the nodes and arcs that are not on any path from s_1^1 to s_n^1. STOP.

Fig. 6. EP: expanding $G_R(V_R, A_R)$ to form $G_E(V_E, A_E)$.

In step 3(v), $\mathbf{y}_j^k$ is inserted into Y_j in lexicographic order so that the search in step 3(iv), which attempts to match $\mathbf{y}_j^k$ with some $\mathbf{y}_j^{k'}$ in Y_j, is efficient. A *lexicographic order* is a *total ordering* in that every two vectors are either equal, or one is lexicographically greater than the other. A nonzero vector is *lexicographically positive* if its first non-zero coordinate is positive.

The vector x is *lexicographically greater than* the vector y if $x - y$ is lexicographically positive, and this defines a *lexicographic order* in $\mathbb{R}^n$. In EP, once all successors of $v_j \in V_1$ are processed, set Y_j can be eliminated, freeing the memory needed to store it.

Step 3(ii) calculates y_{jr}^k using RP; it increases the opportunities to find a vector in Y_j matching $\mathbf{y}_j^k$ in step 3(iv) and, consequently, avoids adding a new node in S_j in step 3(v). All nodes associated with $v_j \in V_R$ that have the same CRR vector are represented by a single node with that CRR vector in G_E. RP reduces the size of G_E. Further, stage 3 of TSA solves SPP on G_E and the computational effort it requires depends on the size of G_E (see the next section); thus, RP reduces the computational effort that would be required by stage 3. The computational evaluation in Chapter VII demonstrates the benefits of RP.

For MRCSP, steps 1-3 may create nodes and arcs that are not on any $s_1^1 - s_n^1$ path, thus, step 4 may be needed to remove such nodes and arcs. Figure 7(a) depicts an example with two resources and $T_1 = T_2 = 2$. After S1A, $G_R = G$. Note that, although arc (1,3) is a bottleneck, S1A does not identify it as such (see Section 5.1). Let G'_E denote the expanded graph that steps 1-3 of EP create. Figure 7(b) shows G'_E, in which node s_5^3 does not have any successor and is not on any $s_1^1 - s_7^1$ path. Thus, for MRCSP, step 4 of EP is needed to delete the nodes and arcs that are not on any $s_1^1 - s_n^1$ path in G'_E. In the example of Figure 7, the subgraph in the dashed box (including arc (s_1^1, s_3^2)) is deleted from G'_E to complete G_E.

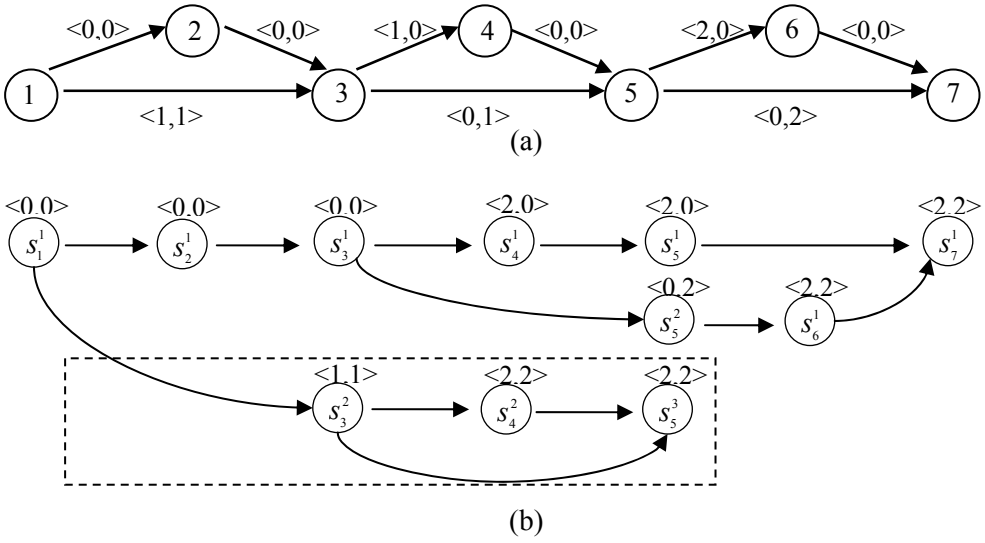
We now present several properties that define the structure of G_E as created by EP.

5.2.2. Properties

Proposition 5.6. One node in G_E corresponds to source node v_1 in G; and another one corresponds to sink node v_n in G; that is $S_1 = \{s_1^1\}$ and $S_n = \{s_n^1\}$.

Proof. Since $\underline{t}_{1r} = \overline{t}_{1r} = 0$ and $\underline{t}_{nr} = \overline{t}_{nr} = T_r$, then $RL_{1r} = \Lambda_{1r} = \{0\}$ and $RL_{nr} = \Lambda_{nr} = \{T_r\}$, $\forall r \in \Re$. Thus, $Y_1 = \{\mathbf{y}_1^1 = \mathbf{0}\}$ and $Y_n = \{\mathbf{y}_n^1 = \mathbf{T}_r\}$, and $S_1 = \{s_1^1\}$ and $S_n = \{s_n^1\}$. ■

By proposition 5.6, s_1^1 is the source node and s_n^1 is the sink node in G_E.



Legend: (a) instance with two resources, the label on each arc is the resource requirement vector; (b) G_E for instance of (a), the label on node s_j^k is vector $\mathbf{y}_j^k$.

Fig. 7. An example of expanded graph G_E.

Proposition 5.7. For SRCSP, every node that steps 1-3 of EP create in G_E is on some path from s_1^1 to s_n^1.

Proof. Consider SRCSP with $\Re = \{1\}$. Note that every node in G_E (except the source node s_1^1) has one or more predecessors because EP constructs G_E in increasing order of $v_j \in V_R$ index. By

way of contradiction, suppose the proposition is not true. Then, steps 1-3 of EP create a node s_j^k that is associated with y_{j1}^k and does not have any successor in G_E, implying $y_{j1}^k > \overline{b}_{j1}$. However, steps 1-3 create node s_j^k only if $y_{j1}^k < \overline{t}_{j1} \le \overline{b}_{j1}$, establishing a contradiction. The proof is completed. ■

By Proposition 5.7, we do not need step 4 of EP for SRCSP.

5.2.3. Computational complexity of EP

To analyze the numbers of nodes and arcs in G_E, define $\vartheta_j = \prod_{r\in\Re} \left|\Lambda_{jr}\right|$ for $v_j \in V_R$ and $\vartheta_{\max} = \max_{v_j \in V_R}\{\vartheta_j\}$. Recall that Λ_{jr} is the set of values of y_{jr}^k for $v_j \in V_R$ and $r \in \Re$. Then, $\left|S_j\right| \le \vartheta_j$ and $\max_{v_j \in V_R}\left|S_j\right| \le \vartheta_{\max}$; that is, the maximal number of nodes associated with any $v_j \in V_R$ is bounded by $\vartheta_{\max}$. Define $\Omega = \sum_{v_j \in V_R} \left|FS(v_j)\right| \vartheta_j$.

Proposition 5.8. The numbers of nodes and arcs in G_E are bounded by $\left|V_R\right| \vartheta_{\max}$ and Ω, respectively, where $\vartheta_{\max} = \max_{v_j \in V_R}\{\vartheta_j\}$ and $\Omega = \sum_{v_j \in V_R} \left|FS(v_j)\right| \vartheta_j$ is of order $O(\left|A_R\right| \vartheta_{\max}) = O(m\vartheta_{\max})$.

Proof. The number of nodes in G_E that are associated with node v_j in G_R is bounded by $\vartheta_j \le \vartheta_{\max}$. Thus, the total number of nodes in G_E is bounded by $\left|V_R\right| \vartheta_{\max}$. Because each node $s_j^k \in S_j \subseteq V_E$ has at most $\left|FS(v_j)\right|$ successors, the total number of arcs in G_E is bounded by $\Omega = \sum_{v_j \in V_R} \left|FS(v_j)\right| \vartheta_j$, which is of order $O\left(\sum_{v_j \in V_R} \left|FS(v_j)\right| \vartheta_{\max}\right) = O(\left|A_R\right| \vartheta_{\max}) = O(m\vartheta_{\max})$. ■

To analyze the complexity of EP, define $\theta_r = \sum_{(i,j)\in A_R} \left|\overline{t}_{jr} - \underline{t}_{jr} + 1\right|$ for $r \in \Re$ and $\Theta = \sum_{r\in\Re} \theta_r$. Recall that $\left|RL_{jr}\right| \le \overline{t}_{jr} - \underline{t}_{jr} + 1$ (see Remark 1). Thus, RL_{jr} for $v_j \in V_R$, $r \in \Re$ can be stored in a

one-dimensional array of size $(\overline{t}_{jr} - \underline{t}_{jr} + 1)$. Using this data structure for RL_{jr}, each $\mathbf{y}_j^k$ can be calculated in $O(|\Re|)$ time according to RP (4-5) in step 3(ii) of EP.

Proposition 5.9. EP runs in $O\left(|\Re| m \vartheta_{\max} + \Theta\right)$ time in the worst-case.

Proof. Constructing sets RL_{jr} for all $v_j \in V_R$ can be done in $O(\theta_r)$ for each $r \in \Re$. Thus, constructing all sets RL_{jr} for $v_j \in V_R$, $r \in \Re$ (step 1) can be done in $O\left(\sum_{r \in \Re} \theta_r\right) = O(\Theta)$ time. Considering arc $(v_i, v_j) \in A_R$, each $\mathbf{y}_j^k$ can be calculated from each $\mathbf{y}_i^h \in Y_i$ in $O(|\Re|)$ time. Because $|Y_i|$ is of order $O(\vartheta_i)$, the calculation of $\mathbf{y}_j^k$ from all $\mathbf{y}_i^h \in Y_i$ can be done in $O(|\Re| \vartheta_i)$ time. Inserting all resulting $\mathbf{y}_j^k$ into Y_j in lexicographic order can be done in $O(|\Re| \vartheta_j)$ time, because vectors $\mathbf{y}_i^h \in Y_i$ are stored and processed in lexicographic order. Thus, step 3 can be done in $O\left(\sum_{(i,j) \in A_R} (|\Re| \vartheta_i + |\Re| \vartheta_j)\right) = O\left(|\Re| m \vartheta_{\max}\right)$. For MRCSP, deleting the nodes and arcs that are not on any connected $s_1^1 - s_n^1$ path in step 4 can be done in $O(\Omega)$ because $|A_E|$ is bounded by Ω by Proposition 5.8. Hence, the total run time of EP is $O\left(|\Re| m \vartheta_{\max} + \Theta\right)$. ■

Remark 2. The solution state (CRR vector) is associated with a feasible combination of resources in set $\Re$. Thus, the number of solution states increases with the number of resources, $|\Re|$. The factors $\vartheta_j = \prod_{r \in \Re} |\Lambda_{jr}|$ for $v_j \in V_R$ and $\vartheta_{\max} = \max_{v_j \in V_R} \{\vartheta_j\}$ in Propositions 5.8 and 5.9 increase quickly with the number of resources. On the other hand, it is likely that more bottleneck nodes and arcs can be identified and deleted to reduce the size of G_R as the number of resources increases. The computational evaluation of Chapter VII investigates this trade-off.

5.2.4. Summary

Via stage 2, SPPRW on G_R is transformed to SPP from source node s_1^1 to sink node s_n^1 on G_E. G_E is acyclic and its nodes are in topological order (i.e., arc (s_i^h, s_j^k) is in G_E if and only if $i < j$); note that no pair of nodes in set of S_j is connected. The cost on arc $(s_i^h, s_j^k) \in A_E$, for $s_i^h \in S_i$, $s_j^k \in S_j$ is the cost on arc $(v_i, v_j) \in A$, c_{ij}.

5.3. Iterative solution stage

Stages 1 (S1A in Figure 3) and 2 (EP in Figure 6) of TSA reduce the original RCSP to SPP on G_E with source node s_1^1 and sink node s_n^1. The task of stage 3 (iterative solution stage) is to optimize (or reoptimize) SPP on acyclic graph G_E. This section proposes an optimizing algorithm (OA) for stage 3 and analyzes its computational complexity. Chapter VIII will propose a reoptimizing algorithm (ROA) that can be used in stage 3 (in place of OA) in CG applications.

Let $FS_E(s_j^k)$ and $BS_E(s_j^k)$ denote the sets of the successors and predecessors of s_j^k in G_E, respectively. $FS_E(s_j^k) = \{s_i^h : (s_j^k, s_i^h) \in A_E\}$ and $BS_E(s_j^k) = \{s_i^h : (s_i^h, s_j^k) \in A_E\}$. Recall that c_{ij} is the cost on arc $(v_i, v_j) \in A$ and arc $(s_i^h, s_j^k) \in A_E$, for $s_i^h \in S_i$, $s_j^k \in S_j$. The cost of a path is the sum of the costs associated with the arcs on the path. Let $\pi(s_j^k)$ be the *label* on node s_j^k, that is, the minimum cost among $s_1^1 - s_j^k$ paths in G_E. Then, the cost of an optimal path is $z = \pi(s_n^1)$ (since $S_n = \{s_n^1\}$ by Proposition 5.6). Let $p(s_j^k)$ be the predecessor of s_j^k on the shortest path from s_1^1 to s_j^k. Given z, a shortest path can be identified easily by backtracing using $p(s_n^1)$. Figure 8 details OA, an algorithm for optimizing SPP on G_E.

step 1. Set $\pi(s_1^1) = 0$.
step 2. For each $S_j \subseteq V_E$ in the order of increasing index, for each $s_j^k \in S_j$, calculate
$\pi(s_j^k) = \min\{\pi(s_i^h) + c_{ij} : (s_i^h, s_j^k) \in A_E\}$ and
$p(s_j^k) = s_{i^*}^{h^*}$ with $\pi(s_j^k) = \pi(s_{i^*}^{h^*}) + c_{i^* j}, s_{i^*}^{h^*} \in BS_E(s_j^k)$.
step 3. Set $z = \pi(s_n^1)$.
step 4. Find the shortest path by tracing back using $p(s_n^1)$. STOP.

Fig. 8. OA.

Proposition 5.10. OA runs in $O(|A_E|) = O(\Omega)$.

Proof. Step 2 of OA processes every arc in G_E in constant time, so it can be done in $O(|A_E|)$ time, where A_E is the arc set of G_E. Similarly, step 4 of OA traces from the sink node back to the source node in $O(|A_E|)$ time. By Proposition 5.8, the number of arcs in G_E is bounded by Ω; thus, OA runs in $O(|A_E|) = O(\Omega)$ time. ■

5.4. TSA for repeatedly solving RCSP

Based on the analysis and results in Sections 5.1-5.3, this section states a version of TSA for solving RCSP repeatedly and analyzes its computational complexity. Figure 9 details TSA, using *iteration* to refer to the CG iterations on which an instance of RCSP is solved repeatedly. Figure 9 shows that the preliminary phase of TSA (stages 1 and 2) is implemented only once in solving an instance of RCSP. When $iteration > 1$, only stage 3 is needed.

1 If $iteration = 1$,
2 (stage 1): run S1A;
3 (stage 2): run EP;
4 (stage 3): run OA on G_E.
5 If $iteration > 1$,
6 (stage 3): run OA on G_E.

Fig. 9. TSA.

Proposition 5.11. The complexity of TSA is $O\left(\left|\Re\right|m\gamma + \left|\Re\right|m\vartheta_{\max} + \Theta\right)$ for the first-time solution ($iteration = 1$). Each subsequent solution ($iteration > 1$) requires $O(\Omega)$ time.

Proof. By Propositions 5.5, 5.9 and 5.10, TSA runs in $O(\left|\Re\right|m\gamma) + O\left(\left|\Re\right|m\vartheta_{\max} + \Theta\right) + O(\Omega)$ $= O\left(\left|\Re\right|m\gamma + \left|\Re\right|m\vartheta_{\max} + \Theta\right)$ time for the first-time solution. Each subsequent solution can be obtained by stage 3 and thus requires $O(\Omega)$ time. ■

The power of TSA is demonstrated when RCSP is solved repeatedly (e.g. in CG). For each iterative solution, only stage 3 is needed and it runs in $O(\Omega) = O(m\vartheta_{\max})$ time. Chapter VII demonstrates the computational effectiveness of TSA when it is used to repeatedly solve RCSP.

5.5. Summary

This chapter proposes TSA and gives time complexities of each stage and of the entire approach. TSA is suitable for solving RCSP repeatedly, for example, when RCSP is a subproblem in CG and CG/B&B. For each iterative solution, only stage 3 of TSA is needed to solve RCSP and it runs in $O(\Omega) = O(m\vartheta_{\max})$ time.

The next chapter presents another method, which may be preferred for one-time solution of RCSP; Chapter VII uses it as a benchmark method for comparing computational results.

CHAPTER VI

LABEL-SETTING ALGORITHM

This chapter presents a label-setting algorithm (LSA) for RCSP. Most previous studies chose various labeling algorithms to solve RCSP subproblem(s) in CG and CG/B&B; thus, this dissertation uses LSA as a benchmark in its computational evaluation of TSA in Chapter VII.

In this chapter, y_{jr}^k is still used to denote CRR-r for $r \in \Re$ for a $v_1 - v_j$ path and $\mathbf{y}_j^k = (y_{j1}^k, \ldots, y_{j|\Re|}^k)$ is the CRR vector. For each node $v_j \in V_R$, let D_j be a set of labels that comprise all pairs $(\tau_j^k, \mathbf{y}_j^k)$ of $v_1 - v_j$ path cost τ_j^k and CRR vector $\mathbf{y}_j^k$ such that $\mathbf{y}_j^k \le \overline{\mathbf{t}}_j$. Then, the recursion of LSA, by increasing order of j, can be expressed as follows.

$$\begin{aligned} D_1 &= \{(\tau_1^1, \mathbf{y}_1^1) : \tau_1^1 = 0, \mathbf{y}_1^1 = \mathbf{0}\}, \\ D_j &= \{(\tau_j^k, \mathbf{y}_j^k) : \tau_j^k = \min\{\tau_i^h + c_{ij} : \text{calculate } \mathbf{y}_j^k \text{ according to (4) and (5)}, \\ &\qquad \mathbf{y}_j^k \le \overline{\mathbf{t}}_j, (\tau_i^h, \mathbf{y}_i^h) \in D_i, (i,j) \in A_R\}\}. \end{aligned} \tag{6}$$

The recursion initializes the cost at the source node of v_1 to be zero and its CRR vector to be the zero vector. For each node v_j, set D_j is calculated by processing every label in D_i for every predecessor v_i of v_j. The recursion continues until set D_n (i.e., label at the sink node of v_n) is obtained. Note that $D_n = \{(\tau_n^1, \mathbf{y}_n^1) : \mathbf{y}_n^1 = \mathbf{T}\}$ since $\underline{t}_{nr} = \overline{t}_{nr} = T_r$ and $RL_{nr} = \{T_r\}$, $\forall r \in \Re$. Thus, the optimal value is

$$z = \tau_n^1. \tag{7}$$

Figure 10 details LSA.

1	If $iteration = 1$,
2	run S1A and construct RL_{jr} for $v_j \in V_R$ and $r \in \Re$ according to (3) in Section 5.2.
3	If $iteration \geq 1$,
4	implement recursion (6) by increasing $v_j \in V_R$ index;
5	calculate the optimal value (expression (7)).

Fig. 10. LSA.

Before solving RCSP for the first time, LSA uses S1A to remove the bottleneck nodes and arcs and formulate resource windows $[\underline{\mathbf{t}}_j, \overline{\mathbf{t}}_j]$, which are used in calculating D_j. Line 2 determines ordered sets RL_{jr} (for $v_j \in V_R$ and $r \in \Re$) for use in calculating y_{jr}^k in recursion (6). For each iteration, recursion (6) of LSA must run from scratch (lines 4-5 in Figure 10). Proposition 6.1 establishes the complexity of LSA.

Proposition 6.1. LSA computes a first-time solution ($iteration = 1$) in $O\left(|\Re| m\gamma + |\Re| m\vartheta_{\max} + \Theta\right)$ in the worst-case. Each subsequent solution ($iteration > 1$) requires $O\left(|\Re| m\vartheta_{\max}\right)$ time.

Proof. By Proposition 5.5, S1A runs in $O\left(|\Re| m\gamma\right)$, where γ is the number of cycles of step 3 in S1A. For SRCSP, $\gamma = 1$. By Proposition 5.9, RL_{jr} for $r \in \Re$ and $v_j \in V_R$ can be determined in $O(\Theta)$ time and each $\mathbf{y}_j^k$ can be calculated in $O(|\Re|)$. $|D_j| \leq \vartheta_j \leq \vartheta_{\max}$, so the total number of labels is bounded by $\Omega = \sum_{v_j \in V_R} |FS(v_j)| \vartheta_j$, which is of order $O(|A_R| \vartheta_{\max}) = O(m\vartheta_{\max})$. Thus, recursion (6) runs in $O\left(|\Re| m\vartheta_{\max}\right)$ time. Hence, the total run time of LSA is $O\left(|\Re| m\gamma + |\Re| m\vartheta_{\max} + \Theta\right)$ for a first-time solution. Each subsequent solution can be obtained by recursion (6), which runs in $O\left(|\Re| m\vartheta_{\max}\right)$ time. ■

By Propositions 5.11 and 6.1, LSA and TSA have the same worst-case complexity for the first-time solution. However, TSA uses a preliminary phase to facilitate each iterative

solution, and, for each subsequent solution of RCSP, TSA requires $O(\Omega) = O(m\vartheta_{\max})$ time while LSA requires $O\left(|\Re| m\vartheta_{\max}\right)$ time. Thus, TSA can be expected to outperform LSA when RCSP is solved repeatedly. Chapter VII explores this conjecture computationally.

CHAPTER VII

COMPUTATIONAL EVALUATION OF TSA

This chapter describes computational tests, designed to evaluate the effectiveness of TSA. Section 7.1 describes test instances and the computation platform. Section 7.2 presents results comparing TSA and LSA on the test instances. Section 7.3 investigates the effect of resource limitations on the performance of TSA. Furthermore, Section 7.4 evaluates the effectiveness of the preliminary phase of TSA.

7.1. Test problems and computational platform

The set of test instances involves acyclic graphs and can be divided into three classes. Class 1 consists of 12 instances from Beasley and Christofides (1989), which are available from the OR-library. Beasley and Christofides (1989) provided 24 test instances, but 12 instances involve cyclic graphs; we use the 12 acyclic graphs, which range from 100 nodes and 959 arcs to 500 nodes and 4,868 arcs. Their instances involve either 1 or 10 resources (see Table 5); they generated resource requirements and arc costs independently from the discrete uniform distribution on range $[0,5]$ (i.e., $DU[0,5]$).

We generate instances in Classes 2 and 3 randomly. Class 2 comprises SRCSP instances; and Class 3, MRCSP instances with *4 resources*. We generated instances in which each arc (i,j) is included in the graph with probability p. In order to assure that the optimal path contains at least n/q arcs, for arc (i,j), j is defined for each integer on $[i+1, \min(n, i+q)]$, where we specified $q\,(1 < q \le n)$ to restrict the span of arc (i,j) so that $j - i \le q$ (see Remark 4 for the definition of span of an arc). The expected number of arcs in a

graph that is randomly generated in this manner with parameters of n, p and q is $pq\left(n-(q+1)/2\right)$.

Class 2 comprises two types of instances (types 2a and 2b) according to the way we assign resource requirements to arcs:

type 2a: the resource requirements on arcs are independent, identically distributed from $DU\left[1,10\right]$;

type 2b: the resource requirement on each arc is positively related to its span, i.e., resource requirement on arc (i, j) equals $[R(j-i)]$ where $[\bullet]$ denotes the nearest integer and R is generated randomly from $U\left(0.0,1.0\right)$.

Class 3 comprises three types of instances (types 3a, 3b, and 3c) according to the way we assign resource requirements to arcs:

type 3a: the requirement for each resource is assigned independently as for type 2a;

type 3b: the requirement for each resource is assigned independently as for type 2b;

type 3c: the requirements for resources 1 to $\left|\Re\right|/2$ ($\left|\Re\right|$ is an even integer) are mutually independently drawn from $DU\left[1,100\right]$. Requirements of resources $r=\left|\Re\right|/2+1,\ldots,\left|\Re\right|$ are inversely related to resource $r-\left|\Re\right|/2$; that is, $u_{ijr}=\left[2500U\left(0.0,1.0\right)/u_{ij.r-\left|\Re\right|/2}\right]$.

Let $T_{\min,r}$ ($T_{\max,r}$) denote the requirements for resource $r\in\Re$ on the v_1-v_n path(s) that require the minimum (maximum) amount of resource r. These values can be obtained by setting $c_{ij}=u_{ijr}$ ($c_{ij}=-u_{ijr}$) and implementing a classical SPP algorithm for each resource $r\in\Re$ on input graph G. For each instance in Classes 2 and 3, the limitation for resource $r\in\Re$ is determined by

$$T_r=T_{\min,r}+(T_{\max,r}-T_{\min,r})\times\eta\,, \tag{8}$$

where we specify parameter η on $(0,1.0]$ to control the tightness of resource limitations. For all instances in Classes 2 and 3, we draw arc costs independently from $U(-100.0, 100.0)$.

Table 4 describes each instance in Classes 2 and 3 in detail. The first column gives a code that identifies each test instance. A triple (*s* or *m*, $|V|$, class type) denotes each instance in Classes 2 and 3, where *s* indicates a single resource, *m* denotes multiple resources, and class type

Table 4. Test instances in Classes 2 and 3

instance	$\|\Re\|$	$\|V\|$	$\|A\|$	q	p	resource 1		resource 2		resource 3		resource 4	
						T_{min}	T_{max}	T_{min}	T_{max}	T_{min}	T_{max}	T_{min}	T_{max}
s-100-2a	1	100	946	25	0.45	10	308	-	-	-	-	-	-
s-100-2b	1					9	98	-	-	-	-	-	-
s-200-2a	1	200	2150	50	0.25	8	446	-	-	-	-	-	-
s-200-2b	1					13	194	-	-	-	-	-	-
s-500-2a	1	500	5450	125	0.1	9	494	-	-	-	-	-	-
s-500-2b	1					38	491	-	-	-	-	-	-
s-700-2a	1	700	6973	140	0.08	10	657	-	-	-	-	-	-
s-700-2b	1					34	684	-	-	-	-	-	-
s-1000-2a	1	1000	10435	240	0.05	14	615	-	-	-	-	-	-
s-1000-2b	1					49	962	-	-	-	-	-	-
s-2000-2a	1	2000	18913	550	0.02	7	470	-	-	-	-	-	-
s-2000-2b	1					116	1926	-	-	-	-	-	-
s-3000-2a	1	3000	29013	750	0.015	10	560	-	-	-	-	-	-
s-3000-2b	1					153	2888	-	-	-	-	-	-
s-5000-2a	1	5000	45038	1000	0.01	11	668	-	-	-	-	-	-
s-5000-2b	1					283	4783	-	-	-	-	-	-
m-20-3a	4	20	155	20	0.8	3	105	6	73	4	82	3	90
m-20-3b	4					3	19	2	19	2	19	2	19
m-20-3c	4					22	807	43	888	20	3082	9	2548
m-50-3a	4	50	472	40	0.4	4	146	6	171	4	155	5	151
m-50-3b	4					5	48	7	48	2	48	7	49
m-50-3c	4					29	1384	23	1414	31	2748	5	4167
m-100-3a	4	100	946	25	0.45	9	253	8	305	10	312	11	282
m-100-3b	4					9	96	11	99	12	99	14	99
m-100-3c	4					74	2785	44	2939	27	6998	38	11668
m-500-3a	4	500	5450	125	0.1	9	525	12	487	13	487	10	515
m-500-3b	4					29	488	33	486	38	481	32	489
m-500-3c	4					86	4630	59	4740	37	18435	30	16337
m-1000-3a	4	1000	10435	240	0.05	10	572	11	576	11	601	10	599
m-1000-3b	4					61	957	54	969	46	951	65	973
m-1000-3c	4					58	5652	72	5673	32	22210	44	24065

is 2a, 2b, 3a, 3b, or 3c. For example, *s*-100-2a denotes the SRCSP instance of class type 2a with 100 nodes; *m*-50-3b denotes the MRCSP instance of class type 3b with 50 nodes. Columns 2 and 3 give $|V|$ and $|A|$, respectively, to record the size of input graph G in each instance. Columns 4 and 5 are the values of parameters q and p, respectively, that are used to generate G. Columns 6 and 7 give $T_{\min,r}$ and $T_{\max,r}$ for $r = 1$ and columns 8-13 give corresponding values for $r = 2,\ 3,$ and 4.

We program all algorithms in the C/C++ programming language and conduct all experiments on a 3.2 GHz Pentium IV PC with 512 Mb of RAM.

7.2. Computational results for TSA and LSA

We solve each instance using both TSA and LSA, each for 100 randomly generated replications for each set of specified resource limitation(s). Stage 3 of TSA uses OA. At each replication, we generate a new set of arc costs randomly from $DU[0,5]$ for instances in Class 1 (to be consistent with the cost structure in Beasley and Christofides (1989)) and from $U(-100.0, 100.0)$ for instances in Classes 2 and 3.

Tables 5, 6, and 7 give computational results for instances in Classes 1, 2 and 3, respectively. In these tables, the column that gives γ reports the number of cycles made by step 3 in S1A. Columns of $|V_E|$ and $|A_E|$ record the size of expanded graph G_E in each instance. The run times for stages 1 and 2 of TSA and for LSA preprocessing are recorded separately; these operations are conducted only once before 100 replications are made. The run times for stage 3 of TSA and for LSA recursion are the total run times for 100 replications (excluding the preliminary phase of TSA and LSA preprocessing, respectively). The last column gives the threshold number of replications, which is calculated using

$$threshold = \left\lceil \frac{100 \times \left(cpu(\text{TSA\_preliminary}) - cpu(\text{LSA\_preprocessing}) \right)}{cpu(\text{LSA\_recursion}) - cpu(\text{stage 3\_of\_TSA})} \right\rceil, \tag{9}$$

where $\lceil \bullet \rceil$ denotes the nearest integer towards infinity. Note that $cpu(\text{stage 3\_of\_TSA})$ is the total run time for stage 3 of TSA for 100 replications and does not include the time for the preliminary phase of TSA; $cpu(\text{LSA\_recursion})$ is the total run time for LSA recursion for 100 replications and does not include LSA preprocessing time. The preliminary phase of TSA requires more run time than LSA preprocessing. However, for each subsequent solution, TSA requires less run time than LSA. Thus, when the number of replications is greater than or equal to *threshold* , TSA is faster than LSA; otherwise, LSA is faster than TSA.

Table 5 presents results of tests on Class 1 instances. Beasley and Christofides (1989) determined the resource limitations for each instance in Class 1. We calculate the average value of η over all of resource types for each instance to provide intuition concerning how tight these resource limitations are. As shown in column 5 of Table 5, the average value of η for all Class 1 instances is less than 0.1, indicating that their limitations are tight. Both TSA and LSA solve all Class 1 instances, but TSA is much faster than LSA over 100 replications. Actually, *threshold* is 3 for all Class 1 instances except rcsp23, which results in a threshold of 5. Note that both instances rcsp23 and rcsp24 have 500 nodes and 10 resources; however, rcsp24 has many more bottleneck nodes (411 for rcsp24 versus 127 for rcsp23) and more bottleneck arcs (4,735 for rcsp24 versus 3,545 for rcsp23) so that rcsp24 is solved more quickly than rcsp23.

We test each Class 2 SRCSP instance using three values of $\eta = \{0.1, 0.5, 0.9\}$, ranging from tight to loose resource limitation(s); instance sizes range from 100 nodes and 946 arcs to 5,000 nodes and 45,038 arcs. Table 6 gives the results of these tests. Columns 2 and 3 record η and T_1, respectively. Other columns correspond to those in Table 5.

TSA requires much less run time than LSA for 100 replications, although stage 2 of TSA may consume some time to expand G_R. Actually, for all Class 2 instances, when the number of replications is larger than or equal to 4, TSA is faster than LSA (see the last column in Table 6). The complexity analysis of Chapters V and VI shows that run times, for TSA and LSA are related, not only to the size of an instance but also to the amounts of resources required by arcs. For example, S1A identifies and deletes the same numbers of bottleneck nodes and arcs in large instances of type 2a (*s*-3000-2a) and type 2b (*s*-3000-2b) with $\eta = 0.5$. TSA solves instance *s*-3000-2a in a run time of 72.089 seconds. However, TSA runs a long time and terminates due to low memory while attempting to solve instance *s*-3000-2b. Because the amounts of resources required in the instance of type 2b are larger than those in the corresponding instance of type 2a; it turns out that TSA takes a long time to generate a much larger expanded graph associated with *s*-3000-2b than that associated with *s*-3000-2a and, eventually, runs out of memory to store such a large graph. Although LSA can solve these instances, it takes 117 seconds for instance *s*-3000-2b for just one replication. Another observation is that the total number of labels enumerated by LSA equals the number of nodes in expanded graph G_E, which is created by TSA. Because TSA and LSA generate the same set of $\mathbf{y}_j^k$; each unique $\mathbf{y}_j^k$ is associated with a node in G_E by TSA and with a label by LSA. Since type 2a instances have arc resource requirements that are uniformly distributed and type 2b instances have arc resource requirements that are proportional to the spans of arcs (so that large values of arc resource requirements may be generated for large span arcs), type 2b instances have more bottleneck arcs than type 2a instances for the same value of η; this difference is especially distinct for $\eta = 0.1$. For all test instances, both the size of G_E and the total run time for $\eta = 0.5$ are much larger than they are for $\eta = 0.1$ or $\eta = 0.9$. The next section investigates this issue in detail.

Table 5. Results of solving instances in Class 1 for 100 replications

instance	$\lvert\Re\rvert$	$\lvert V\rvert$	$\lvert A\rvert$	avg η	γ	NBN	NBA	$\lvert V_E\rvert$	$\lvert A_E\rvert$	TSA run time (sec.) stage1	stage2	stage3	total	NL	LSA run time (sec.) prep	recursion	total	*threshold*
rcsp3	1	100	959	0.08	1	3	24	1016	7662	0	0.203	0.094	0.297	1016	0.015	8.516	8.531	3
rcsp4				0.07	1	3	32	849	5962	0	0.156	0.078	0.234	849	0.015	6.750	6.765	3
rcsp7	10		999	0.10	1	10(28)*	160(779)**	419	594	0	1.156	0.047	1.203	4490	0.156	36.485	36.641	3
rcsp8				0.08	3	11(82)	293(974)	31	40	0	0.328	0.032	0.360	1070	0.11	7.640	7.750	3
rcsp11	1	200	1971	0.10	1	7	56	3744	33961	0	0.844	0.359	1.203	3744	0.047	35.75	35.797	3
rcsp12				0.09	1	7	56	3262	28866	0	0.703	0.313	1.016	3262	0.047	30.797	30.844	3
rcsp15	10		1960	0.07	2	20(136)	259(1848)	153	199	0	2.047	0.047	2.094	8364	0.296	63.954	64.250	3
rcsp16				0.06	3	24(191)	362(1950)	9	10	0	0.562	0.047	0.609	1933	0.219	13.219	13.438	3
rcsp19	1	500	4978	0.06	1	28	296	6992	59336	0	1.453	0.516	1.969	6992	0.11	64.063	64.173	3
rcsp20				0.05	1	28	310	5739	416158	0	1.188	0.422	1.610	5739	0.078	50.767	50.845	3
rcsp23	10		4868	0.07	1	31(127)	309(3545)	3515	4755	0	107.767	0.390	108.16	231622	0.969	2504.87	2505.84	5
rcsp24				0.06	2	31(411)	379(4735)	142	187	0	9.454	0.047	9.501	36998	0.797	305.94	306.737	3

NBN – number of bottleneck nodes

NBA – number of bottleneck arcs

NL – number of labels generated by LSA

prep - LSA preprocessing

*: number of bottleneck vertices identified by S1A; the term in parentheses gives total number of bottleneck nodes

**: number of bottleneck arcs identified by S1A; the term in parentheses gives total number of bottleneck arcs

Table 6. Results of solving instances in Class 2 for 100 replications

instance	η	T_1	γ	NBN	NBA	$\|V_E\|$	$\|A_E\|$	TSA run time (sec.)				NL	LSA run time (sec.)			*threshold*
								stage1	stage2	stage3	total		prep	recursion	total	
s-100-2a	0.1	39	1	6	71	1797	15306	0	0.406	0.171	0.577	1797	0.031	18.16	18.181	3
	0.5	159	1	6	71	5868	61395	0	1.406	0.423	1.829	5868	0.094	65.31	65.404	3
	0.9	278	1	6	71	1154	11956	0	0.265	0.126	0.391	1154	0.016	12.19	12.206	3
s-100-2b	0.1	17	1	7	406	525	2078	0	0.078	0.046	0.124	525	0.016	4.22	4.236	2
	0.5	53	1	6	71	1985	18465	0	0.469	0.187	0.656	1985	0.031	22.19	22.221	2
	0.9	89	1	6	71	703	6976	0	0.156	0.095	0.251	703	0.015	8.75	8.765	2
s-200-2a	0.1	51	1	9	94	5595	56093	0	1.422	0.329	1.751	5595	0.093	64.07	64.163	3
	0.5	227	1	9	94	18927	226843	0	5.125	1.188	6.313	18927	0.328	240.94	241.268	2
	0.9	402	1	9	94	5126	60318	0	1.156	0.248	1.404	5126	0.031	57.19	57.221	2
s-200-2b	0.1	31	1	10	790	2149	10489	0	0.360	0.158	0.518	2149	0.016	16.25	16.266	3
	0.5	103	1	9	94	7925	85128	0	2.000	0.546	2.546	7925	0.125	97.03	97.155	2
	0.9	175	1	9	94	2802	31652	0	0.718	0.188	0.906	2802	0.031	34.54	34.571	2
s-500-2a	0.1	57	1	45	610	15005	152044	0	3.703	0.958	4.661	15005	0.265	172.04	172.305	3
	0.5	251	1	45	610	44861	526350	0	11.703	2.907	14.61	44861	0.750	562.36	563.11	2
	0.9	445	1	45	610	9485	111743	0	2.156	0.562	2.718	9485	0.063	106.09	106.153	2
s-500-2b	0.1	83	1	47	2215	11605	57040	0	1.781	0.455	2.236	11605	0.125	77.03	77.155	3
	0.5	264	1	45	610	44532	469984	0	12.032	2.767	14.799	44532	0.750	521.88	522.63	3
	0.9	445	1	45	610	11105	125874	0	2.625	0.671	3.296	11105	0.125	128.75	128.875	2
s-700-2a	0.1	74	1	47	517	30444	293955	0	7.234	1.798	9.032	30444	0.500	324.53	325.03	3
	0.5	333	1	47	517	93872	1008892	0.015	23.953	5.674	29.642	93872	1.469	1072.66	1074.129	3
	0.9	592	1	47	517	20128	216881	0	4.344	0.970	5.314	20128	0.171	203.29	203.461	3
s-700-2b	0.1	99	1	47	2050	25444	135816	0.016	4.109	0.892	5.017	25444	0.281	177.35	177.631	3
	0.5	359	1	47	517	94782	943704	0	26.157	5.561	31.718	94782	1.468	1036.42	1037.888	3
	0.9	619	1	47	517	25828	272210	0	5.672	1.322	6.994	25828	0.296	272.82	273.116	2
s-1000-2a	0.1	74	1	115	1362	38269	378134	0	8.985	2.392	11.377	38269	0.625	421.72	422.345	2
	0.5	314	1	115	1362	111872	1239197	0	27.422	6.938	34.36	111872	1.719	1308.72	1310.439	2
	0.9	554	1	115	1362	21040	239205	0	4.610	1.096	5.706	21040	0.156	225.31	225.466	2

Table 6. Continued

instance	η	T_1	γ	NBN	NBA	$\lvert V_E \rvert$	$\lvert A_E \rvert$	TSA run time (sec.)				NL	LSA run time (sec.)			*threshold*
								stage1	stage2	stage3	total		prep	recursion	total	
s-1000-2b	0.1	140	1	116	4602	37379	177380	0	5.657	1.233	6.890	37379	0.391	226.09	226.481	3
	0.5	505	1	115	1362	173035	1761164	0	50.454	10.90	61.354	173035	2.719	1940.79	1943.509	3
	0.9	870	1	115	1362	40907	447040	0	9.969	2.095	12.064	40907	0.469	441.10	441.569	3
s-2000-2a	0.1	53	1	372	4037	34371	277175	0	7.203	1.955	9.158	34371	0.547	339.06	339.607	2
	0.5	238	1	372	3970	146380	1508221	0	35.454	8.953	44.407	146380	2.172	1585.94	1588.112	3
	0.9	423	1	372	3970	12988	130094	0	2.797	0.583	3.3795	12988	0.109	148.28	148.389	2
s-2000-2b	0.1	297	1	425	11778	35371	121304	0	4.719	0.938	5.6565	35371	0.532	175.62	176.152	3
	0.5	1021	1	372	3970	503833	4731109	0	174.833	29.535	204.37	503833	7.766	5185.975	5193.741	4
	0.9	1745	1	372	3970	49082	514541	0	10.563	2.583	13.146	49082	0.672	507.97	508.642	2
s-3000-2a	0.1	65	1	559	6062	78963	705546	0.016	17.343	4.535	21.894	78963	1.219	811.27	812.489	2
	0.5	285	1	559	6060	248658	2591222	0	57.014	15.075	72.089	248658	3.516	2700.075	2703.591	2
	0.9	505	1	559	6060	9503	94517	0	2.281	0.230	2.511	9503	0.078	136.25	136.328	2
s-3000-2b	0.1	426	1	603	16817	90781	352494	0	13.39	2.750	16.14	90781	1.516	461.73	463.246	3
	0.5	1520	1	559	6060	1126408	10827859	0.015	625.927	low memory		1126408	17.438	11709.75	11727.19	-
	0.9	2614	1	559	6060	75915	828642	0.015	16.844	4.145	21.004	75915	1.094	809.08	810.174	2
s-5000-2a	0.1	76	1	1042	10511	152924	1293007	0	33.704	8.775	42.479	152924	2.266	1577.35	1579.616	3
	0.5	339	1	1042	10510	509976	4920653	0	126.034	29.993	156.03	509976	7.063	5196.90	5203.963	3
	0.9	602	1	1042	10510	6431	61231	0	2.016	0.510	2.526	6431	0.047	153.900	153.947	2
s-5000-2b	0.1	733	1	1068	22086	392417	1752358	0	81.19	13.55	94.74	392417	5.260	2118.375	2123.635	4
	0.5	2533	1	1042	10510	3161210	29054297	0	3804.74	low memory		3161210	47.735	31446.3	31494.04	-
	0.9	4333	1	1042	10510	154419	1585921	0.016	34.22	8.710	42.946	154419	2.500	1542.975	1545.475	3

Table 7 gives the results on MRCSP instances with 4 resources; underlying graphs range from 20 nodes and 155 arcs to 1,000 nodes and 10,435 arcs. Columns 3-6 give the resource limitations for resource $r = 1$, 2, 3, and 4, respectively. Other columns correspond to those in Tables 5 and 6.

Both TSA and LSA solve small instances (e.g., *m*-20 series (i.e., *m*-20-3a, *m*-20-3b and *m*-20-3c)) in reasonable times. They also solve mid-size instances (e.g., *m*-50 and *m*-100 series) for both small and large values of η but their performances degrade with the mid-range value of η. On large instances (e.g., *m*-500 and *m*-1000 series), we test only small and large values of η because both TSA and LSA take long times to solve each instance with the mid-range value of η. This is consistent with the complexity analyses of Chapters V and VI. The performances of TSA and LSA dis-improve on MRCSP instances because the size of the solution space increases with the number of resources (see Remark 2). Note that, for MRCSP instances, stage 2 of TSA identifies some bottleneck nodes and arcs that stage 1 of TSA can not identify, as illustrated in Section 5.1.

Comparing *s*-100-2a versus *m*-100-3a and *s*-100-2b versus *m*-100-3b, we see that, with multiple resources, more bottleneck nodes and arcs can be identified and deleted if η is small (e.g., $\eta = 0.1$), but as η increases, the multiple resources do not help identify more bottleneck nodes and arcs (e.g., $\eta = 0.5$ and $\eta = 0.9$).

In summary, consistent with our TSA and LSA complexity analysis, the run times required by TSA and LSA depend not only on the size of the underlying graph but also on the number of resources and the amounts of resources required by arcs. The run time required by stage 1 (i.e., S1A) is negligible for almost all of test instances; S1A is indeed efficient. The number of cycles in S1A, γ, is at most 6 over all MRCSP instances and $\gamma = 1$ for all SRCSP

Table 7. Results of solving instances in Class 3 for 100 replications

instance	η	T_1	T_2	T_3	T_4	γ	NBN	NBA	$\|V_E\|$	$\|A_E\|$
m-20-3a	0.1	13	12	11	11	1	17	152	3	3
	0.5	54	39	43	46	1	1	12	1970	5721
	0.9	94	66	74	81	1	1	12	265	1407
m-20-3b	0.1	infeasible			-	-	1	-	-	-
	0.5	11	10	10	10	4	2(9)	104(142)	11	14
	0.9	17	17	17	17	1	1	14	523	2349
m-20-3c	0.1	100	127	326	262	1	11	141	8	14
	0.5	414	465	1551	1278	1	1	14	1675	6549
	0.9	728	803	2775	2294	1	1	12	142	1017
m-50-3a	0.1	18	22	19	19	4	22(26)	405(423)	34	60
	0.5	75	88	79	78	1	5	83	244860	794024
	0.9	131	154	139	136	1	5	38	2209	14222
m-50-3b	0.1	infeasible			-	-	1	-	-	-
	0.5	26	27	25	28	1	5	163(205)	9617	17167
	0.9	43	43	43	44	1	5	84	4364	24295
m-50-3c	0.1	164	162	302	421	3	36	447	15	27
	0.5	706	718	1389	2086	1	5	85	114474	467701
	0.9	1248	1274	2476	3750	1	5	83	1413	9101
m-100-3a	0.05	21	22	25	24	4	93	938	7	8
	0.1	33	37	40	38	1	6	71(110)	26223	44139
	0.9	228	275	281	254	1	6	71	666279	6162433
	0.95	240	290	296	268	1	6	71	6403	62057
m-100-3b	≤0.2	infeasible			-	1	-	-	-	-
	0.25	30	33	33	35	2	6(42)	143(825)	388	492
	0.9	low memory			-	-	-	-	-	-
	0.95	91	94	94	94	1	6	71	44096	369813
m-100-3c	0.05	200	188	375	619	6	22(63)	619(880)	76	104
	0.1	345	333	724	1201	1	6	91(94)	201708	377057
	0.9	2513	2649	6300	10505	1	6	71	554514	5109023
	0.95	2649	2794	6649	11086	1	6	71	6042	56656
m-500-3a	0.05	34	35	36	35	2	66(152)	1136(3829)	5330	8050
	0.95	499	463	463	489	1	45	610	211742	2393833
m-500-3b	0.05	infeasible			-	1	-	-	-	-
	0.95	465	463	458	466	1	45	610	270554	2915987
m-500-3c	0.05	313	293	956	845	3	55(97)	1111(2642)	34349	49637
	0.95	4402	4505	17515	15521	1	45	610	50110	631858
m-1000-3a	0.05	38	39	40	39	2	139(263)	1964(6841)	14934	21375
	0.95	543	547	571	569	1	115	1362	1240084	13725213
m-1000-3b	0.05	infeasible			-	1	-	-	-	-
	0.95	low memory			-	-	-	-	-	-
m-1000-3c	0.05	337	352	1140	1245	2	134(160)	1929(3331)	179.837	260.888
	0.95	5372	5392	21101	22863	1	115	1362	250552	3090572

Table 7. Continued

instance	TSA run time (sec.)				NL	LSA run time (sec.)			*threshold*
	stage1	stage2	stage3	total		prep	recursion	total	
m-20-3a	0	0	0.031	0.031	3	0	0.46	0.46	0
	0	0.531	0.063	0.594	2093	0	21.56	21.56	3
	0	0.078	0.063	0.141	267	0	4.22	4.22	2
m-20-3b	-	-	-	-	-	-	-	-	-
	0	0	0.048	0.048	27	0	0.46	0.46	0
	0	0.156	0.063	0.219	530	0	10.16	10.16	2
m-20-3c	0	0	0.078	0.078	9	0	1.343	1.343	0
	0	0.609	0.093	0.702	1680	0.015	33.344	33.359	2
	0	0.046	0.048	0.094	143	0	3.031	3.031	2
m-50-3a	0	0.016	0.016	0.032	43	0	1.671	1.671	1
	0	3795.9	6.687	3802.61	258093	0.031	176730.4	176730	3
	0	0.719	0.171	0.89	2213	0	40.969	40.969	2
m-50-3b	-	-	-	-	-	-	-	-	-
	0	17.125	0.328	17.453	28061	0	399.69	399.69	5
	0	2.484	0.282	2.766	4370	0	178.13	178.13	2
m-50-3c	0	0.015	0.047	0.062	16	0	1.719	1.719	1
	0	959.8	3.344	963.147	116503	0.156	62289.8	62290	2
	0	0.5	0.14	0.64	1415	0.03	28.61	28.64	2
m-100-3a	0	0.015	0.061	0.076	7	0	1.72	1.72	1
	0	64.797	0.578	65.375	77368	0.094	1513.3	1513.39	5
	0	1963.9	157.25	2121.19	666448	0.046	82676.1	82676.1	3
	0	3.157	0.375	3.532	6404	0.015	186.25	186.265	2
m-100-3b	-	-	-	-	-	-	-	-	-
	0	178.86	0.141	179.002	153969	0.063	2553.15	2553.21	8
	-	-	-	-	-	-	>10 hrs	-	-
	0	77.61	1.971	79.581	44097	0.015	6284.45	6284.47	2
m-100-3c	0	0.219	0.048	0.267	334	0.172	2.781	2.953	2
	0	2858.2	4.654	2862.89	338557	1.797	100335.3	100337	3
	0	2995.9	27.218	3023.16	554515	0.296	132839.2	132839	3
	0	2.86	0.36	3.22	6043	0.063	177.19	177.253	2
m-500-3a	0	5.813	0.157	5.97	29379	0.344	210.16	210.504	3
	0	155.56	11.608	167.172	211743	0.094	8550.9	8550.99	2
m-500-3b	-	-	-	-	-	-	-	-	-
	0	999.66	16.079	1015.74	270555	0.14	84829.2	84829.3	2
m-500-3c	0	66.391	0.687	67.078	96445	8.781	2088.47	2097.25	3
	0	38.203	2.766	40.969	50111	0.266	2536.59	2536.86	2
m-1000-3a	0	25.25	0.501	25.751	98642	0.75	877.35	878.1	3
	0	3458.2	19516	22974.7	1240085	0.188	105121.7	105122	5
m-1000-3b	-	-	-	-	-	-	-	-	-
	-	-	-	-	1401506	0.439	1045992	1045992	-
m-1000-3c	0	908.22	4.38	912.595	592414	24.516	26429.2	26453.7	4
	0	263.14	16.03	279.173	250554	0.563	17473.65	17474.2	2

instances. For all test instances, the run time of TSA stage 3 is proportional to the size of G_E, and the run time of LSA is proportional to total number of labels enumerated. TSA and LSA both solve RCSP effectively; their performances on SRCSP are better than on MRCSP. TSA takes advantage of its preliminary phase, which is implemented only once for each instance to make G_E available to facilitate subsequent calculation. In contract, LSA must run its recursion from the beginning for each reoptimization. TSA significantly outperforms LSA on all test instances that involve repetitive solutions. TSA is most suitable for applications that use RCSP as a subproblem in CG or CG/B&B, while LSA is preferred for one-time solution.

7.3. Effect of resource limitations on TSA

To demonstrate the effect of resource limitations on TSA performance, we conduct a series of tests on selected instances: *s*-500 series and *m*-50 series with $\eta = \{ 0.1,\ldots,1.0 \}$. Tables 8 and 9 give computational results on *s*-500 series and *m*-50 series, respectively. Columns in Tables 8 and 9 correspond to those in Tables 6 and 7, respectively.

To exemplify performance, Figures 11(a) and 11(b) depict the size of G_E and the run times of stages 2 and 3 as a function of η in instance *s*-500-2a, respectively. They demonstrate that both the size of G_E and run time increase with η but then decrease as η continues to increase. Each curve has a single peak occurring around the median value of η. Specifically, the peak occurs at $\eta = 0.5$ for *s*-500-2a, *s*-500-2b, *m*-50-3a, and *m*-50-3c, and at $\eta = 0.7$ for *m*-50-3b. When η takes the mid-range value, the size of G_E is large and the run times of stages 2 and 3 are long. If η is small (e.g., 0.1), many bottleneck nodes and arcs can be identified and deleted and few feasible $v_1 - v_n$ paths exist. On the other hand, if η is large (e.g., 0.9), some

paths are always feasible with respect to resource limitations and G_E includes them directly without expanding them. The extreme case is that $\eta = 1.0$, for which $G_E = G_R$.

Table 8. Results of solving the *s*-500 series for $\eta = \{0.1,\ldots,1.0\}$

instance	η	T_1	NBN	NBA	$\lvert V_E\rvert$	$\lvert A_E\rvert$	TSA run time (sec.)			
							stage1	stage2	stage3	total
s-500-2a	0.1	57	45	610	15005	152044	0	3.703	0.958	4.661
	0.2	106	45	610	28778	319643	0	7.547	1.891	9.438
	0.3	154	45	610	38237	438197	0	9.656	2.560	12.216
	0.4	203	45	610	43551	507481	0	11.422	2.765	14.187
	0.5	251	45	610	44861	526350	0	11.703	2.907	14.61
	0.6	300	45	610	42179	495970	0	10.86	2.625	13.485
	0.7	348	45	610	35070	412773	0	8.375	2.109	10.484
	0.8	397	45	610	23467	275838	0	5.391	1.360	6.751
	0.9	445	45	610	9485	111743	0	2.156	0.562	2.718
	1	494	45	610	455	4840	0	0.171	0.064	0.235
s-500-2b	0.1	83	47	2215	11605	57040	0	1.781	0.455	2.236
	0.2	128	45	896	25744	197345	0	5.437	1.412	6.849
	0.3	173	45	615	36141	333601	0	8.938	2.108	11.046
	0.4	219	45	610	42486	427600	0.016	11.453	2.517	13.986
	0.5	264	45	610	44532	469984	0	12.032	2.767	14.799
	0.6	309	45	610	42468	461333	0	11.421	2.612	14.033
	0.7	355	45	610	36108	399201	0.015	9.031	2.055	11.101
	0.8	400	45	610	25494	285003	0	6.140	1.575	7.715
	0.9	445	45	610	11105	125874	0	2.625	0.671	3.296
	1	491	45	610	455	4840	0	0.171	0.064	0.235

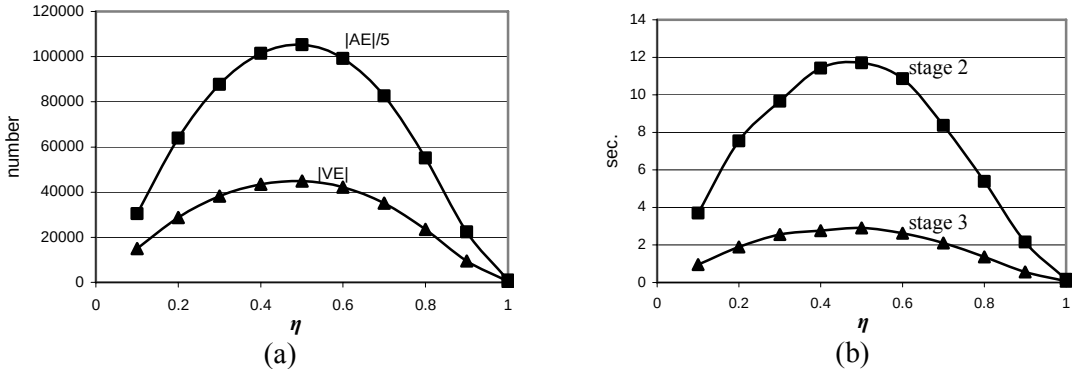


Fig. 11. Instance *s*-500-2a.

Table 9 shows that, for MRCSP instances, the size of G_E and run time to solve it are quite sensitive to η. For example, relative to instance *m*-50-3a, as η increases from 0.1 to 0.5, the size of G_E increases from 34 nodes and 60 arcs to 244,860 nodes and 794,024 arcs and the run time of stage 2 increases from 0.016 second to 3,795.9 seconds; as η increases further from 0.5 to 0.9, the size of G_E decreases to 2,209 nodes and 14,222 arcs and run time of stage 2 decreases to 0.719 second.

Table 9. Results of solving the *m*-50 series for $\eta = \{0.1,\ldots,1.0\}$

instance	η	T_1	T_2	T_3	T_4	NBN	NBA	$\lvert V_E\rvert$	$\lvert A_E\rvert$	TSA run time (sec.)			
										stage1	stage2	stage3	total
m-50-3a	0.1	18	22	19	19	22(26)	405(423)	34	60	0	0.016	0.016	0.032
	0.2	32	39	34	34	5	89(98)	2120	4886	0.015	0.656	0.032	0.703
	0.3	46	55	49	48	5	83	28735	74615	0	52.063	0.733	52.796
	0.4	60	72	64	63	5	83	131222	383115	0	1134.17	3.391	1137.56
	0.5	75	88	79	78	5	83	244860	794024	0	3795.93	6.687	3802.61
	0.6	89	105	94	92	5	83	216061	779777	0.015	2743.45	6.267	2749.74
	0.7	103	121	109	107	5	83	97518	392873	0	359.926	2.685	362.611
	0.8	117	138	124	121	5	83	22647	105654	0	14.875	0.720	15.595
	0.9	131	154	139	136	5	83	2209	14222	0.015	0.719	0.171	0.905
	1	146	171	155	151	5	83	45	389	0	0.016	0.077	0.093
m-50-3b	0.1	9	11	6	11	50	472	infeasible		-	-	-	-
	0.2	13	15	11	15	50	472	infeasible		-	-	-	-
	0.3	17	19	15	19	32(35)	444(452)	22	29	0	0.016	0.062	0.078
	0.4	22	23	20	23	5(6)	223(345)	491	726	0	0.406	0.078	0.484
	0.5	26	27	25	28	5	163(205)	9617	17167	0	17.125	0.328	17.453
	0.6	30	31	29	32	5	125(142)	78421	173544	0	533.972	1.908	535.88
	0.7	35	35	34	36	5	108	229676	630848	0	3889.15	5.828	3894.98
	0.8	39	39	38	40	5	92	89096	331482	0	520.20	2.406	522.606
	0.9	43	43	43	44	5	84	4364	24295	0	2.484	0.282	2.766
	1	48	48	48	49	5	83	45	389	0	0.031	0.063	0.094
m-50-3c	0.1	164	162	302	421	36	447	15	27	0	0.015	0.047	0.062
	0.2	300	301	574	837	5	97(105)	1793	4426	0	0.688	0.062	0.75
	0.3	435	440	846	1253	5	87	22514	67667	0	32.875	0.639	33.514
	0.4	571	579	1117	1669	5	86	82604	289443	0	474.903	2.156	477.059
	0.5	706	718	1389	2086	5	85	114474	467701	0	959.803	3.344	963.147
	0.6	842	857	1661	2502	5	83	91493	415852	0	556.887	2.563	559.45
	0.7	977	996	1932	2918	5	83	43332	213512	0	105.671	1.373	107.044
	0.8	1113	1135	2204	3334	5	83	8377	50011	0	5.500	0.437	5.937
	0.9	1248	1274	2476	3750	5	83	1413	9101	0	0.500	0.140	0.64
	1	1384	1414	2748	4167	5	83	44	389	0	0.016	0.126	0.142

7.4. Effectiveness of prescribing resource windows and RP in TSA

The goal of this section is to demonstrate the effectiveness of prescribing resource windows in stage 1 and calculating $\mathbf{y}_j^k$ with RP in stage 2 of TSA. For this purpose, we adapt EP in Subsection 7.4.1 to devise an expanded graph approach (EGA). EGA neither prescribes resource windows and nor uses RP to calculate $\mathbf{y}_j^k$. We use the *s*-500 and *m*-20 series to compare TSA and EGA computationally in Subsection 7.4.2.

7.4.1. Description of EGA

Let $G_{EGA} = (W, E)$ be the expanded graph created by EGA ($G_E(V_E, A_E)$ denotes the expanded graph created by TSA). Let W_j be the set of all nodes $w_j^k \in W$ for $k = 1, \ldots, |W_j|$ that are associated with a specific $v_j \in V_R$ and E_{ij} be the set of all arcs $(w_i^h, w_j^k) \in E$ associated with a given arc $(i, j) \in A_R$. EGA constructs an expanded graph in a manner similar to that used by EP, but using resource windows $[\mathbf{0}, \mathbf{T}]$ at each node and calculating y_{jr}^k according to

$$y_{jr}^k = y_{ir}^h + u_{ijr}, \text{ for } (w_i^h, w_j^k) \in E. \tag{10}$$

That is, EGA does not prescribe tight resource windows as stage 1 of TSA (S1A) does, and it calculates $\mathbf{y}_j^k$ without RP.

A sink node w_{n+1}^1 must be included in W and corresponding arcs (w_n^h, w_{n+1}^1), for $w_n^h \in W_n$ must be included in E because W_n contains more than one node. Arc (w_i^h, w_j^k) is in E only if $\mathbf{y}_j^k = \mathbf{y}_i^h + \mathbf{u}_{ij} \le \mathbf{T}$, where $\mathbf{y}_i^h$ and $\mathbf{y}_j^k$ are CRR vectors associated with w_i^h and w_j^k, respectively. EGA can construct G_{EGA} based on the following recursion, which processes nodes $v_j \in V_R$ in increasing index order:

$$\begin{aligned} W_1 &= \{w_1^1 : \mathbf{y}_1^1 = 0\}, \\ W_j &= \{w_j^k : \mathbf{y}_j^k = \mathbf{y}_i^h + \mathbf{u}_{ij} : \mathbf{y}_j^k \le \mathbf{T}, w_i^h \in W_i, (v_i, v_j) \in A_R\}, \\ E_{ij} &= \{(w_i^h, w_j^k) : \mathbf{y}_j^k = \mathbf{y}_i^h + \mathbf{u}_{ij} : \mathbf{y}_j^k \le \mathbf{T}, w_i^h \in W_i, w_j^k \in W_j\}, \\ E_{n,n+1} &= \{(w_n^h, w_{n+1}^1) : w_n^h \in W_n\}. \end{aligned} \tag{11}$$

To make a fair comparison, the network-reduction technique (steps 1-4 of S1A) is used in conjunctive with EGA to identify and delete bottleneck nodes and arcs before constructing G_{EGA}. After constructing G_{EGA} as in (11), the nodes and arcs that are not on any path from w_1^1 to w_{n+1}^1 are deleted to complete G_{EGA}. The optimal value (i.e., minimum cost) and optimal path can be found using OA. Based on Propositions 5.5, 5.9 and 5.10, the following proposition is established.

Proposition 7.1. The number of arcs in G_{EGA} is bounded by $O(m\varphi)$. The expanding stage of EGA constructs G_{EGA} in $O(|\Re| m\varphi)$, where $\varphi = \prod_{r=1}^{|\Re|}(T_r + 1)$. The minimum cost and the optimal path can be computed in $O(|E|) = O(m\varphi)$ time. The total run time for a first-time solution ($iteration = 1$) is $O(|\Re| m\gamma + |\Re| m\varphi)$, where term $O(|\Re| m\gamma)$ represents the time to identify and delete bottleneck nodes and arcs before constructing G_{EGA}. Each subsequent solution ($iteration > 1$) requires $O(m\varphi)$ time.

Proof. Because $|W_j| = O(\varphi)$ and each node w_j^k in G_{EGA} has at most $|FS(v_j)|$ successors, the number of arcs in G_{EGA} is bounded by $O\left(\sum_{v_j \in V_R} |FS(v_j)| \varphi\right) = O(m\varphi)$. Considering arc $(v_i, v_j) \in A_R$, $\mathbf{y}_j^k$ can be calculated from each $\mathbf{y}_i^h$ for $w_i^h \in W_i$ in $O(|\Re|)$ time. That is, each node w_j^k (associated with $\mathbf{y}_j^k$) can be created in $O(|\Re|)$ time. To facilitate the test if a duplicate of $\mathbf{y}_j^k$ already exist, we store CRR vectors $\mathbf{y}_j^k$ for $w_j^k \in W_j$ in lexicographic order for $v_j \in V_R$. Because

$|W_i| = O(\varphi)$, the calculation of $\mathbf{y}_j^k$ from all $\mathbf{y}_i^h$ for $w_i^h \in W_i$ can be done in $O(|\Re|\varphi)$ time. Further, resulting $\mathbf{y}_j^k$ are in lexicographic order because $\mathbf{y}_i^h$ for $w_i^h \in W_i$ are stored and processed in that order. Thus, inserting all resulting nodes w_j^k (associated with $\mathbf{y}_j^k$) into W_j ($|W_j| = O(\varphi)$) requires $O(|\Re|\varphi)$ time to check if duplicates of $\mathbf{y}_j^k$ already exists; simultaneously, all resulting $\mathbf{y}_j^k$ can be inserted into established $\mathbf{y}_j^k$ for $w_j^k \in W_j$ in lexicographic order. Thus, the expanding stage of EGA constructs G_{EGA} in $O\left(\sum_{(i,j)\in A_R}(|\Re|\varphi)\right) = O(|\Re|m\varphi)$ time. Deleting nodes and arcs that are not on any $w_1^1 - w_{n+1}^1$ path can be done in $O(m\varphi)$ time. By Proposition 5.10, the minimum cost and the optimal path can be computed in $O(|E|) = O(m\varphi)$ time. Thus, the total run time for a first-time solution is $O(|\Re|m\gamma + |\Re|m\varphi)$, where term $O(|\Re|m\gamma)$ represents the time to identify and delete bottleneck nodes and arcs before constructing G_{EGA} (see Proposition 5.5). Each subsequent solution can be found using OA in $O(m\varphi)$ time. ■

Recall that TSA runs in $O(|\Re|m\gamma + |\Re|m\vartheta_{\max} + \Theta)$ time for the first-time solution; each subsequent solution can be obtained by stage 3 and requires $O(\Omega) = O(m\vartheta_{\max})$ time (see Proposition 5.10). Note that, in general, $\Theta = \sum_{r\in\Re}\theta_r = \sum_{r\in\Re}\sum_{(i,j)\in A_R}(\bar{t}_{jr} - \underline{t}_{jr} + 1) << \sum_{r\in\Re}\sum_{(i,j)\in A_R}(T_r + 1)$ $= m\sum_{r\in\Re}(T_r + 1) << |\Re|m\varphi$, and $\vartheta_j = \prod_{r=1}^{|\Re|}|\Lambda_{jr}| << \prod_{r=1}^{|\Re|}(\bar{t}_{jr} - \underline{t}_{jr} + 1) << \varphi$ for $v_j \in V_R$, so $\vartheta_{\max} << \varphi$. Thus, TSA offers much better worst-case performance than EGA. Since EGA does not prescribe tight resource windows for nodes in G_R, it generates many nodes and arcs that eventually are not on any path from w_1^1 to w_{n+1}^1 and must be deleted to complete G_{EGA}. Further, EGA calculates $\mathbf{y}_j^k$ using (10) without RP (4-5); thus, it enumerates more distinct values of vectors $\mathbf{y}_j^k$ and

consequently generates more nodes in G_{EGA}. The shape of G_{EGA} can be imagined as a pyramid; and the shape of G_E as a barrel. These points are demonstrated by the following computational results (see Table 10).

7.4.2. Computational results

Table 10 shows that G_{EGA} has many more nodes and arcs than G_E for every instance. It turns out that G_{EGA} includes many duplicated partial paths that are not necessary. The density of G_{EGA} ($|W|/|E|$) is almost 100 percent, indicating that EGA merges few nodes as it constructs G_{EGA}. Since the run time for solving SPP on an expanded graph (either G_{EGA} or G_E) in stage 3 depends on the size of expanded graph, the run time for solving an instance on G_{EGA} is longer than that on G_E. Another observation is that the size of G_{EGA} increases with η, while the size of G_E increases with η to a peak and then decreases as η continues to increase (see Section 7.3). As η increases, the performance of EGA degrades further and further. For example, EGA solves instance *m*-20-3a with $\eta = 0.9$ in 121.56 seconds, generating G_{EGA} with 44,086 nodes and 44,259 arcs, but TSA can easily solve this instance in 0.14 second, generating G_E with 265 nodes and 1,407 arcs. In summary, by prescribing resource windows and using RP to calculate $\mathbf{y}_j^k$, TSA expands the partial paths in G_R if and only if necessary; thus, TSA is more effective than EGA. Hence, prescribing resource windows and calculating $\mathbf{y}_j^k$ using RP are effective and especially important for TSA.

Table 10. Comparison of TSA and EGA

instance	η	TSA		run time (sec.)				EGA:		run time (sec.)			
		$\|V_E\|$	$\|A_E\|$	stage1	stage2	stage3	total	$\|W\|$	$\|E\|$	stage1	stage2	stage3	total
s-500-2a	0.1	15005	152044	0	3.703	0.958	4.661	20102	187416	0	4.235	1.232	5.467
	0.5	44861	526350	0	11.70	2.907	14.61	78737	772990	0	16.407	4.689	21.096
	0.9	9485	111743	0	2.156	0.562	2.718	103217	952418	0	21.078	5.719	26.797
s-500-2b	0.1	11605	57040	0	1.781	0.455	2.236	23863	131770	0	3.547	0.877	4.424
	0.5	44532	469984	0	12.03	2.767	14.799	76625	706352	0	18.516	4.550	23.066
	0.9	11105	125874	0	2.625	0.671	3.296	96532	892090	0	24.344	5.301	29.645
m-20-3a	0.1	3	3	0	0	0.031	0.031	3	3	0	0	0.031	0.031
	0.5	1970	5721	0	0.531	0.063	0.594	10083	10127	0	10.657	0.188	10.845
	0.9	265	1407	0	0.078	0.063	0.141	44086	44259	0	120.93	0.625	121.56
m-20-3b	0.1	infeasible		-	-	-	-	-	-	-	-	-	-
	0.5	12	14	0	0	0.048	0.048	14	14	0	0.016	0.048	0.064
	0.9	523	2349	0	0.156	0.063	0.219	17161	20677	0	23.235	0.251	23.486
m-20-3c	0.1	8	14	0	0	0.078	0.078	14	14	0	0.015	0.078	0.093
	0.5	1675	6549	0	0.609	0.093	0.702	17142	17142	0	26.828	0.297	27.125
	0.9	142	1017	0	0.046	0.048	0.094	44551	44551	0	91.408	0.515	91.923

CHAPTER VIII

TSA FOR SOLVING RCSP IN CG AND CG/B&B

In this chapter, we consider the special issues that arise when RCSP is used as subproblem(s) in CG and CG/B&B. In the context of CG, arc costs are updated using the new values of dual variables at each CG iteration and the subproblem must be reoptimized with respect to these new arc costs. Section 8.1 proposes ROA that can be used in stage 3 of TSA in CG applications and presents a version of TSA for solving RCSP subproblem using ROA in CG (TSA-CG). Further, in the context of CG/B&B, some arcs (i, j) in graph G may be forbidden or prescribed (i.e., associated decision variables x_{ij} that are fixed to 0 or 1, respectively, by the branching rule). Section 8.2 proposes a method for dealing with these fixed arcs (MDFA) and presents a version of TSA for solving RCSP as a subproblem in CG/B&B (TSA-CG/B&B).

8.1. ROA and TSA-CG

When RCSP is used as a subproblem in CG, the RCSP subproblem must be reoptimized with respect to the new arc costs at each CG iteration. Of course, OA (Figure 8) can be used to find a (new) shortest path with respect to the updated arc costs. This section proposes an alternative method – ROA (Subsection 8.1.1) and analyzes the complexity of ROA (Subsection 8.1.2). Based on that, Subsection 8.1.3 presents TSA-CG and Subsection 8.1.4 analyzes the complexity of TSA-CG. Finally, Subsection 8.1.5 shows the computational tests on ROA.

8.1.1. Description of ROA

Rather than solving from scratch to prescribe each iterative solution, ROA only updates the labels $\pi(s_j^k)$ that are affected by the new arc costs, using the shortest path tree found at the last

iteration (i.e., using the previous arc costs). A shortest path tree comprises the shortest path from source node s_1^1 to each of the other nodes in G_E (i.e., $V_E \setminus \{s_1^1\}$).

Let B be a set of arcs that have updated (new) costs. These new costs may be smaller or larger than the old ones. Let c_{ij} denote the old cost on arc $(v_i, v_j) \in A$; and c'_{ij}, the new cost on the arc. Correspondingly, let $\pi(s_j^k)$ denote the minimum cost among $s_1^1 - s_j^k$ paths at the last iteration; and $\pi'(s_j^k)$, the minimum cost at the current iteration. Recall that $p(s_j^k)$ denotes the predecessor of s_j^k in the shortest path tree. Let H be a heap that stores a set of arcs in G that have updated costs; and H_E, a heap that stores a set of arcs in G_E, in which the tail of each arc has updated value of $\pi(s_j^k)$. Using this notation, Figure 12 details ROA (with respect to B).

ROA is a label-setting algorithm, which is tailored for RCSP on an acyclic graph. Initialize heap H with the arcs in B (step 1) and the process continues until $H = \varnothing$ and $H_E = \varnothing$ (step 2). On each iteration of ROA, the arc with the smallest index j from either H or H_E is selected for preprocessing (step 2(i)). If the selected arc is from H, then steps 2(ii-x) apply; otherwise, steps 2(xi-xviii) apply. Because arc (i, j) in G might correspond to several arcs in G_E, each with tail in S_i and head in S_j, respectively, the processing of arc (i, j) in H involves dealing with all arcs (s_i^h, s_j^k) in G_E that correspond to (i, j) (see step 2(iv)). In steps 2(v-viii), each arc (s_i^h, s_j^k) that corresponds to arc (i, j) in H is processed as follows (j here is j_1 in ROA of Figure 12). If $c'_{ij} > c_{ij}$ and $p(s_j^k) \neq s_i^h$ (step 2(v)), the current shortest path (i.e., the shortest path prescribed at the last iteration) is still optimal. If $c'_{ij} > c_{ij}$ and $p(s_j^k) = s_i^h$ (step 2(vi)), then $\pi'(s_j^k)$ and $p(s_j^k)$ are updated to be $\min\{\pi'(s_i^h) + c'_{ij} : s_i^h \in BS_E(s_j^k)\}$ and $\arg\min_{s_i^h \in BS_E(s_j^k)} \{\pi'(s_i^h) + c'_{ij}\}$, respectively. If $c'_{ij} < c_{ij}$ and $p(s_j^k) \neq s_i^h$ (step 2(vii)), and, in addition, if

step 1. Set $H \leftarrow B$, $H_E = \varnothing$, and $\pi'(s_j^k) = \pi(s_j^k)$, $\forall s_j^k \in V_E$.

step 2. While $H \neq \varnothing$ or $H_E \neq \varnothing$,

(i) let $j_1 = \min\{j : \exists (v_i, v_j) \in H\}$ and $j_2 = \min\{j : \exists (s_i^h, s_j^k) \in H_E\}$.

(ii) If $j_1 \leq j_2$:

(iii) for each $(v_i, v_{j_1}) \in H$,

(iv) for each $s_{j_1}^k \in S_{j_1}$ such that $\exists s_i^h \in BS_E(s_{j_1}^k)$,

(v) if $c'_{ij_1} > c_{ij_1}$ and $p(s_{j_1}^k) \neq s_i^h$, go to step 2(iv);

(vi) if $c'_{ij_1} > c_{ij_1}$ and $p(s_{j_1}^k) = s_i^h$,

$$\pi'(s_{j_1}^k) = \min\{\pi'(s_i^h) + c'_{ij_1} : s_i^h \in BS_E(s_{j_1}^k)\} = \pi'(s_{i^*}^{h^*}) + c'_{i^* j_1};\ p(s_{j_1}^k) = s_{i^*}^{h^*};$$

(vii) if $c'_{ij_1} < c_{ij_1}$ and $p(s_{j_1}^k) \neq s_i^h$, if $\pi'(s_i^h) + c'_{ij_1} < \pi'(s_{j_1}^k)$, $\pi'(s_{j_1}^k) = \pi'(s_i^h) + c'_{ij_1}$;

$p(s_{j_1}^k) = s_i^h$;

(viii) if $c'_{ij_1} < c_{ij_1}$ and $p(s_{j_1}^k) = s_i^h$, $\pi'(s_{j_1}^k) = \pi(s_{j_1}^k) + c'_{ij_1} - c_{ij_1}$;

(ix) if $\pi'(s_{j_1}^k) \neq \pi(s_{j_1}^k)$, insert $\{(s_{j_1}^k, s_i^h) : s_i^h \in FS_E(s_{j_1}^k)\}$ into H_E;

(x) remove (v_i, v_{j_1}) from H.

(xi) If $j_2 \leq j_1$:

(xii) for each $(s_i^h, s_{j_2}^k) \in H_E$,

(xiii) if $\pi'(s_i^h) > \pi(s_i^h)$ and $p(s_{j_2}^k) \neq s_i^h$, do nothing;

(xiv) if $\pi'(s_i^h) > \pi(s_i^h)$ and $p(s_{j_2}^k) = s_i^h$,

$$\pi'(s_{j_2}^k) = \min\{\pi'(s_i^h) + c'_{ij_2} : s_i^h \in BS_E(s_{j_2}^k)\} = \pi'(s_{i^*}^{h^*}) + c'_{i^* j_2};\ p(s_{j_2}^k) = s_{i^*}^{h^*};$$

(xv) if $\pi'(s_i^h) < \pi(s_i^h)$ and $p(s_{j_2}^k) \neq s_i^h$, if $\pi'(s_i^h) + c'_{ij_2} < \pi'(s_{j_2}^k)$, $\pi'(s_{j_2}^k) = \pi'(s_i^h) + c'_{ij_2}$;

$p(s_{j_2}^k) = s_i^h$;

(xvi) if $\pi'(s_i^h) < \pi(s_i^h)$ and $p(s_{j_2}^k) = s_i^h$, $\pi'(s_{j_2}^k) = \pi'(s_i^h) + c'_{ij_2}$;

(xvii) if $\pi'(s_{j_2}^k) \neq \pi(s_{j_2}^k)$, insert $\{(s_{j_2}^k, s_i^h) : s_i^h \in FS_E(s_{j_2}^k)\}$ into H_E;

(xviii) remove $(s_i^h, s_{j_2}^k)$ from H_E.

step 3. Set $z = \pi'(s_n^1)$.

step 4. Find the shortest path by tracing back using $p(s_n^1)$. STOP.

Fig. 12. ROA.

$\pi'(s_i^h) + c'_{ij} < \pi'(s_j^k)$, then $\pi'(s_j^k) = \pi'(s_i^h) + c'_{ij}$ and $p(s_j^k) = s_i^h$; otherwise, the current shortest path is still optimal. If $c'_{ij} < c_{ij}$ and $p(s_j^k) = s_i^h$ (step 2(viii)), then the current shortest path is still optimal but the optimal cost on the path is decreased by amount of $c_{ij} - c'_{ij}$. If $p(s_j^k)$ is changed

during the processing of (s_i^h, s_j^k), all arcs outgoing from s_j^k are inserted into H_E (step 2(ix)). After it is processed, arc (v_i, v_j) is removed from H (step 2(x)). A similar analysis applies to steps 2(xiii-xvii), where $\pi(s_i^k)$ differs from $\pi'(s_i^k)$, but this part of ROA is not detailed to conserve space.

8.1.2. Computational complexity of ROA

ROA processes each node in S_i before any node in S_j for $j > i$. Further, because no pair of nodes in set S_i is connected, heap H_E can be stored as an *n*-dimensional array in which each element is a linked-list of arcs (unsorted). All arcs with head in S_i are inserted into linked-list $H_E[i]$ if labels (i.e., the minimum cost $\pi(s_j^k)$) on their tails are changed by ROA (step 2(ix) and (xvii)). With this heap structure H_E, heap operations (extraction in steps 2(i) and (x) and insertion in steps 2(ix) and (xvii)) can be done in constant time. ROA processes arcs in $H_E[i]$ before processing arcs in $H_E[j]$ for $j > i$. The following proposition establishes the complexity of ROA.

Proposition 8.1. ROA runs in $O(|A_E|) = O(\Omega)$ time in the worst-case.

Proof. Since each operation on heap H_E takes a constant amount of time, the processing of each arc in heap H_E is in constant time. In the worst-case, all arcs in G_E must be processed once; thus, the total run time of ROA is $O(|A_E|) = O(\Omega)$ in the worst-case. ■

Note that OA and ROA are two methods for solving SPP in stage 3 of TSA. They both have the same worst-case complexity of $O(\Omega)$. Since ROA must maintain a heap, the implicit coefficient, c_{ROA}, of Ω is larger than the one, c_{OA}, that is implicit in the expression that describes the complexity of OA. That is, the worst-case complexity of ROA is $O(c_{ROA}\Omega)$ and

that of OA is $O(c_{OA}\Omega)$ with $c_{ROA} > c_{OA}$ so that, ROA may require large run time, on average. On the other hand, the actual run time of ROA relates to set B. In general, the run time of ROA increases with $|B|$. For small $|B|$, ROA is more likely to be faster than OA. Section 8.3 presents computational tests that compare ROA and OA.

8.1.3. Description of TSA-CG

Figure 13 details TSA-CG that incorporates ROA in stage 3 to solve RCSP in CG (without fixed arcs), using *iteration* to refer to a CG iteration. Figure 13 shows that the preliminary phase of TSA (stages 1 and 2) is implemented only once in solving an instance of RCSP. When $iteration > 1$, only stage 3 is needed. λ is a parameter specified by the analyst. If the number of arc costs that are assigned new values on an iteration is greater than λ, SPP is solved from scratch using OA; otherwise, SPP is reoptimized using ROA. $\lambda = 0$ implies OA is implemented on every iteration. Chapter VII designs experiments to estimate an appropriate value for λ for the test instances.

```
1    If iteration = 1,
2        (stage 1):  run S1A;
3        (stage 2):  run EP;
4        (stage 3):  run OA.
5    If iteration > 1,
6        (stage 3):  If |B| ≥ λ, run OA; else (i.e., |B| < λ), run ROA.
```

Fig. 13. TSA-CG.

Remark 3. In implementing TSA to solve RCSP subproblem(s) in CG, the preliminary phase of TSA can help reduce the size of the overall problem. Let $\Im$ denote a problem that can be solved by CG (or CG/B&B) with RCSP subproblem(s). If the preliminary phase identifies arc (i, j) as a bottleneck, it fixes decision variable x_{ij} to 0. All fixed variables can be removed from problem $\Im$. Thus, the preliminary phase of TSA can reduce the size of $\Im$.

8.1.4. Computational complexity of TSA-CG

Proposition 8.2. The worst-case complexity of TSA-CG is $O\left(|\Re| m\gamma + |\Re| m\vartheta_{\max} + \Theta\right)$ for the first-time solution ($iteration = 1$). Each subsequent solution ($iteration > 1$) requires $O(\Omega)$ time.

Proof. Because the complexities of ROA and OA are the same, TSA-CG has the same complexity as TSA in Figure 9, which is $O\left(|\Re| m\gamma + |\Re| m\vartheta_{\max} + \Theta\right)$ for the first-time solution. After that, Each subsequent solution can be obtained by either OA or ROA and thus requires $O(\Omega)$ time. ■

TSA-CG is suitable for repeatedly solving RCSP as a subproblem in CG. At each CG iteration, only stage 3 is needed to solve an RCSP subproblem and it runs in $O(\Omega) = O(m\vartheta_{\max})$ time.

8.1.5. ROA tests

This subsection investigates ROA (Figure 12) by comparing it with OA (Figure 8). Recall that B is the set of arcs in G that are assigned new arc costs. Since stage 2 of TSA expands G, the number of arcs in the expanded graph G_E that are assigned new costs may be much larger than $|B|$. Our experiment tests four values of $|B|$ (1, 3, 5 and 10) on two selected series, *s*-500 and *s*-1000. We run 100 replications for each value of $|B|$ on each instance with a specified $\eta = \{0.1, 0.5, 0.9\}$. We number the arcs in G from 1 to $|A|$. For each replication, we choose $|B|$ arcs at random from G by generating $|B|$ unique integers (corresponding to $|B|$ arcs) from $DU[1, |A|]$. For each selected arc, *e*, we generate a random value Δ from $U(-100.0, -1.0) \cup (1.0, 100.0)$ and add it to the current cost of arc *e*. If Δ is negative, the cost of arc e decreases; otherwise, it increases.

Table 11 presents computational results. The run times in Table 11 do not include the

time for the preliminary phase of TSA because OA and ROA are used only in stage 3 of TSA. Note that the run time for OA does not depend on set B; the run times for OA in column 6 are the average run times per replication over 100 replications. However, the run time for ROA depends on the number of arcs in B and their locations in the graph. Table 11 demonstrates this, giving the average, standard deviation, minimum, and maximum values, of run times for 100 replications (columns 7-10). The last column in Table 11 gives the ratio of the average run time for one replication of ROA to one of OA.

Columns 7-10 in Table 11 show that ROA run time differs significantly as a function of the arcs in set B, even for the same value of $|B|$. For all test instances, the minimum run time of ROA can be 0 second if the arcs in B do not alter the current shortest path tree. However, the maximum run time of ROA can be large, much larger than OA if the arcs in B affect a large portion of G_E (especially if the arcs in B affect the entire graph). Consequently, the standard deviations (column 8) are large, even much larger than average run times of ROA.

Furthermore, Table 11 shows that, on average, if $|B| \leq 3$, ROA is faster than OA, except for instance *s*-500-2b with $\eta = 0.9$, which has the smallest expanded graph (9,484 nodes and 11,173 arcs) among all test instances. However, on average, if $|B| \geq 5$, ROA is slower than OA for most instances except for *s*-1000-2b with $\eta = 0.1,\ 0.5,$ or 0.9. These results are consistent with expectations. Basically, ROA may save time by resolving only portions of G_E that are effected by the arcs in B. In general, the portion of the graph that is affected increases with $|B|$, and may even encompass the entire graph. On the other hand, ROA must maintain an additional heap to store the portion of G_E that has been affected and heap operations require additional run time. This trade-off balances at some value of $|B|$, on average. For most of the test instances, the value of $|B|$ balances at 3~5.

Table 11. Comparison of OA and ROA

instance	η	$\lvert V_E\rvert$	$\lvert A_E\rvert$	$\lvert B\rvert$	OA run time	ROA run time ($\times 10^{-3}$sec.)				ratio
					($\times 10^{-3}$sec.)	mean	stdev	min	max	
s-500-2a	0.1	15004	152044	1	10.32	2.02	6.88	0	47	0.196
				3	10.31	2.96	6.528	0	31	0.287
				5	10.62	10.93	24.993	0	171	**1.029**
				10	10.31	17.66	33.586	0	235	1.713
	0.5	44860	526350	1	30.01	9.85	55.121	0	531	0.328
				3	30.15	24.53	79.526	0	531	0.814
				5	29.99	93.28	268.432	0	1656	**3.110**
				10	30.15	152.02	310.131	0	1688	5.042
	0.9	9484	11173	1	6.87	1.08	3.958	0	16	0.157
				3	6.88	7.18	34.411	0	250	**1.044**
				5	6.72	18.43	59.442	0	344	2.743
				10	6.72	29.69	62.745	0	344	4.418
s-500-2b	0.1	11604	57040	1	5.47	1.09	3.995	0	16	0.199
				3	6.09	1.88	5.602	0	32	0.309
				5	5.01	5.62	10.798	0	63	**1.122**
				10	5.15	7.84	12.08	0	63	1.522
	0.5	44531	469984	1	28.59	7.82	47.27	0	469	0.274
				3	28.59	15.62	52.036	0	359	0.546
				5	28.76	59.84	168.558	0	1078	**2.081**
				10	28.59	116.87	232.379	0	1094	4.088
	0.9	11104	125874	1	7.97	1.72	6.251	0	47	0.216
				3	7.82	3.82	6.766	0	31	0.488
				5	7.96	10.15	35.046	0	296	**1.275**
				10	7.97	19.53	38.727	0	188	2.450
s-1000-2a	0.1	38268	378134	1	24.22	1.87	5.548	0	31	0.077
				3	24.54	7.65	22.729	0	157	0.312
				5	24.06	25.63	101.882	0	922	**1.065**
				10	24.06	39.53	82.868	0	719	1.643
	0.5	111871	1239797	1	70.62	8.29	50.864	0	500	0.117
				3	70.16	29.21	107.244	0	781	0.416
				5	71.25	97.81	332.752	0	2282	**1.373**
				10	72.34	251.42	562.565	0	2765	3.476
	0.9	21039	239205	1	12.19	1.88	7.437	0	62	0.154
				3	12.03	5.31	19.904	0	172	0.441
				5	12.34	17.35	71.108	0	563	**1.406**
				10	12.18	44.54	90.218	0	484	3.657
s-1000-2b	0.1	37378	177380	1	13.6	1.25	4.798	0	31	0.092
				3	13.14	3.44	7.863	0	31	0.262
				5	12.19	9.06	19.116	0	125	0.743
				10	12.03	12.81	20.488	0	109	**1.065**
	0.5	173034	1761164	1	109.53	15	75.526	0	656	0.137
				3	109.54	28.13	79.775	0	703	0.257
				5	110.47	103.9	357.198	0	2844	0.941
				10	109.38	349.07	742.203	0	3485	**3.191**

Table 11. Continued

instance	η	$\lvert V_E\rvert$	$\lvert A_E\rvert$	$\lvert B\rvert$	OA run time ($\times 10^{-3}$sec.)	ROA run time ($\times 10^{-3}$sec.)				ratio
						mean	stdev	min	max	
s-1000-2b	0.9	40906	447040	1	22.66	2.02	6.518	0	47	0.089
				3	21.73	9.53	20.153	0	140	0.439
				5	22.35	16.09	48.137	0	422	0.720
				10	21.87	56.88	93.649	0	437	**2.601**

8.2. MDFA and TSA-CG/B&B

When RCSP is used as a subproblem in CG that is incorporated in a B&B scheme, some arcs (i, j) in the graph may be fixed. Solving RCSP with fixed arcs poses opportunities to specialize TSA to gain effectiveness.

Let F_0 (F_1) be the set of arcs that correspond to the binary variables fixed to 0 (1) at a node in the B&B tree. Then, F_0 is the set of the forbidden arcs on G which are not allowed on the optimal path; and F_1 is the set of the prescribed arcs that must be on the optimal path. At each B&B node F_0 and F_1 are fixed and RCSP subproblem(s) is(are) solved at each CG iteration. Thus, it is worth taking some time to revise G_E based on F_0 and F_1 before solving RCSP subproblem(s). The revised graph is smaller than G_E and the resulting problem can be solved using either OA (Figure 8) or ROA (Figure 12) on the revised graph.

This section contains five subsections. Subsection 8.2.1 investigates properties related to forbidden (F_0) and prescribed arcs (F_1) and proposes MDFA to exploit them. Subsection 8.2.2 presents TSA-CG/B&B, a version of TSA for solving RCSP in CG/B&B, and Subsection 8.2.3 analyzes its complexity. Subsection 8.2.4 presents a computational evaluation of MDFA. Finally, Subsection 8.2.5 summarizes this section and discusses some issues related to implementing TSA-CG/B&B.

8.2.1. Description of MDFA

To avoid confusion, we assume that the preliminary phase of TSA deletes bottleneck arcs from G. Define *implied forbidden arcs* in G as the arcs that can not be on connected $v_1 - v_n$ paths that contain all arcs in F_1 and do not contain any arc in F_0. All arcs in G_E corresponding to forbidden (including implied forbidden) arcs in G can be removed from G_E.

Theorem 8.1. If arc (i, j) in acyclic graph G is prescribed (i.e., x_{ij}=1), then any arc (i', j') with (a) $i' < i$ and $j' > i$, or (b) $i' = i$ and $j' \neq j$, or (c) $i < i' < j$ is an implied (type 1) forbidden arc.

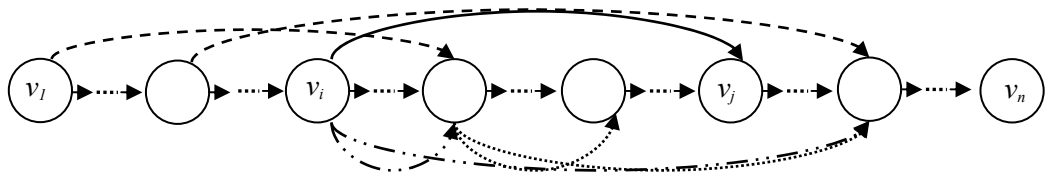
Proof. If arc (i, j) is prescribed, then any arc (i', j') identified by cases (a), (b) and (c) can not be on a $v_1 - v_n$ path together with arc (i, j) (see Figure 14). ■

By the above theorem, the following corollary is straightforward.

Corollary 8.1. Suppose $F_1 \neq \varnothing$, if prescribed arcs in F_1 can not be sorted as

$$\left\{(v_{i_k}, v_{j_k}), k = 1, 2, \ldots \mid F_1 \mid\right\} \text{ such that } i_k < j_k \text{ and } j_k \leq i_{k+1}, \tag{12}$$

then, RCSP subject to the set of prescribed arcs F_1 is infeasible because prescribed arcs conflict.



—— prescribed arc (i, j)

- - - - - (a) arcs (i', j') with $i' < i$ and $j' > i$ are implied forbidden arcs.

—·—·— (b) arcs (i', j') with $i' = i$ and $j' \neq j$ are implied forbidden arcs.

..... (c) arcs (i', j') with $i < i' < j$ are implied forbidden arcs.

Fig. 14. Dashed arcs ((a), (b) and (c)) are forbidden due to prescribed arc (i, j).

By invoking Theorem 8.1, additional implied (type 2) forbidden arcs may be found. After removing implied (type 1) forbidden arcs (i', j') in cases (a), (b) and (c) for each $(i, j) \in F_1$ and arcs in F_0 from G, some of the remaining nodes and arcs may no longer be on any $v_1 - v_n$ path, and so become implied (type 2) forbidden arcs. Let $\hat{F}_0$ be a set of all of forbidden arcs including i) forbidden arcs in F_0; ii) implied (type 1) forbidden arcs stated in Theorem 8.1 for each $(i, j) \in F_1$: $\bigcup_{(i,j)\in F_1} \left\{ \left(\bigcup_{i<i'<j} (i', j') \right) \cup \left(\bigcup_{i'=i, j'\neq j} (i', j') \right) \cup \left(\bigcup_{i'<i, j'>i} (i', j') \right) \right\}$ in which the three terms in parentheses are associated with cases (a), (b) and (c), respectively; iii) implied (type 2) forbidden arcs that are not on any $v_1 - v_n$ path after removing the forbidden arcs in i) and ii). Given F_0 and F_1, the algorithm detailed in Figure 15 – GFA – generates $\hat{F}_0$ in $O(m)$ time.

step 1. Set $\hat{F}_0 = \bigcup_{(i,j)\in F_1} \left\{ \left(\bigcup_{i<i'<j} (i', j') \right) \cup \left(\bigcup_{i'=i, j'\neq j} (i', j') \right) \cup \left(\bigcup_{i'<i, j'>i} (i', j') \right) \right\} \cup F_0$.
For $1 \le j \le n$,
$BS'(v_j) = \left\{ v_i \in BS(v_j) : (v_i, v_j) \notin \hat{F}_0 \right\}$ and $FS'(v_j) = \left\{ v_i \in FS(v_j) : (v_j, v_i) \notin \hat{F}_0 \right\}$.
step 2. For j in increasing order from 2 to n,
if $BS'(v_j) = \varnothing$ and $FS'(v_j) \neq \varnothing$,
$BS'(v_i) = BS'(v_i) \setminus \{v_j\}, \forall v_i \in FS'(v_j)$; $\hat{F}_0 = \hat{F}_0 \cup \left\{ (v_j, v_i) : v_i \in FS'(v_j) \right\}$; $FS'(v_j) = \varnothing$.
step 3. For j in decreasing order from n-1 to 1,
If $FS'(v_j) = \varnothing$ and $BS'(v_j) \neq \varnothing$,
$FS'(v_i) = FS'(v_i) \setminus \{v_j\}, \forall v_i \in BS'(v_j)$; $\hat{F}_0 = \hat{F}_0 \cup \left\{ (v_i, v_j) : v_i \in BS'(v_j) \right\}$; $BS'(v_j) = \varnothing$.
step 4. If any $(i, j) \in F_1$ has $BS'(v_i) = \varnothing$ or $FS'(v_j) = \varnothing$, STOP;
the problem is infeasible because the remaining graph is disconnected.
step 5. Return $\hat{F}_0$.

Fig. 15. GFA: generating $\hat{F}_0$ algorithm.

Step 1 of GFA initializes $\hat{F}_0$ with arcs in F_0 and implied forbidden arcs of type 1. Steps 2 and 3 identify implied forbidden arcs of type 2. Step 4 checks to determine if the remaining graph, denoted as G_{RF}, is connected. Note that, if G_{RF} is connected, every prescribed arc in G_{RF}

is a bridge, that is, an arc such that the connected graph becomes disconnected if it is removed from the graph. Thus, removing implied forbidden arcs is equivalent to fixing prescribed arcs.

Now, based on $\hat{F}_0$, the algorithm detailed in Figure 16 – GERA – can identify and remove all forbidden arcs in G_E. Let G_{EF} be the resulting graph, that is, G_{EF} is formed from G_E by removing all forbidden arcs in G_E. G_{EF} is a subgraph of G_E and we call it the revised graph. Let $FS'_E(s_j^k)$ be a subset of $FS_E(s_j^k)$ that denotes a set of successors in G_{EF}; and $BS'_E(s_j^k)$ be a subset of $BS_E(s_j^k)$ that denotes a set of predecessors in G_{EF}. Note that arc $(i,j) \in \hat{F}_0$ implies that arcs (s_i^h, s_j^k) with $s_i^h \in S_i$ and $s_j^k \in S_j$ are forbidden in G_E. GERA removes all forbidden arcs of G_E that correspond to arcs in $\hat{F}_0$ (step 1). Then, it identifies and removes the arcs in the remaining graph that can not be on any $s_1^1 - s_n^1$ path (steps 2 and 4). If $BS'_E(s_n^1) = \varnothing$ (step 3) or $FS'_E(s_1^1) = \varnothing$ (step 5), G_{EF} is disconnected so STOP; otherwise, return G_{EF} defined by $BS'_E(s_j^k)$ and $FS'_E(s_j^k)$, for $s_j^k \in V_E$ (step 6). Figure 16 details this procedure.

step 1. Set $BS'_E(s_j^k) = BS_E(s_j^k)$, $FS'_E(s_j^k) = FS_E(s_j^k)$, $\forall s_j^k \in V_E$.
For each arc $(s_i^h, s_j^k) \in V_E$ ∋ $(i,j) \in \hat{F}_0$,
$BS'_E(s_j^k) = BS'_E(s_j^k) \setminus \{s_i^h\}$ and $FS'_E(s_i^h) = FS'_E(s_i^h) \setminus \{s_j^k\}$.
step 2. For each $s_j^k \in V_E \setminus \{s_1^1\}$ in increasing j index order from 2 to n,
if $BS'_E(s_j^k) = \varnothing$ and $FS'_E(s_j^k) \neq \varnothing$,
$BS'_E(s_i^h) = BS'_E(s_i^h) \setminus \{s_j^k\}$, $\forall s_i^h \in FS'_E(s_j^k)$; $FS'_E(s_j^k) = \varnothing$.
step 3. If $BS'_E(s_n^1) = \varnothing$, STOP. The problem is infeasible because G_{EF} is disconnected.
step 4. For each $s_j^k \in V_E \setminus \{s_n^1\}$, j in decreasing order from $n-1$ to 1,
if $FS'_E(s_j^k) = \varnothing$ and $BS'_E(s_j^k) \neq \varnothing$,
$FS'_E(s_i^h) = FS'_E(s_i^h) \setminus \{s_j^k\}$, $\forall s_i^h \in BS'_E(s_j^k)$; $BS'_E(s_j^k) = \varnothing$.
step 5. If $FS'_E(s_1^1) = \varnothing$, STOP. The problem is infeasible because G_{EF} is disconnected.
step 6. Return $BS'_E(s_j^k)$ and $FS'_E(s_j^k)$, $\forall s_j^k \in V_E$.

Fig. 16. GERA: G_E revising algorithm.

After revising G_E to G_{EF}, either OA or ROA can be applied to G_{EF} to find an optimal solution by replacing $FS_E(s_j^k)$ with $FS'_E(s_j^k)$ and $BS_E(s_j^k)$ with $BS'_E(s_j^k)$, respectively, in Figures 8 and 12. In addition, ROA replaces B with $B \setminus \hat{F}_0$ because arcs in $\hat{F}_0$ do not appear in G_{EF}.

Remark 4. Define the span of arc (i, j) as $j - i$, assuming that the nodes are topologically numbered. Based on Theorem 8.1, it is appropriate to select a branching variable x_{ij} whose corresponding arc (i, j) has a large span, so that it will generate more implied forbidden arcs and cause G_{EF} to be smaller than if the branching variable has a small span.

8.2.2. Description of TSA-CG/B&B

Figure 17 details TSA-CG/B&B, which incorporates MDFA in the first iteration of CG at each B&B node (except the root node). At the root node of the B&B tree ($F_1 = F_0 = \varnothing$), RCSP is solved using TSA-CG in Figure 13 (line 1). At each other node in the B&B tree some arcs are fixed. The first CG iteration (line 2 of Figure 17), sorting F_1 according to expression (12) (see Theorem 8.1) (line 3) if $F_1 \neq \varnothing$. GFA then generates $\hat{F}_0$ (line 4), and GERA revises G_E to form G_{EF} (line 5) before OA optimizes SPP on G_{EF} (line 6). MDFA is the method that deals with fixed arcs as specified in lines 3-5. For subsequent iterations ($iteration > 1$), only stage 3, which applies either OA or ROA (line 8), is needed. Note that the preliminary phase (involved in TSA-CG in line 1) is conducted only once for the entire problem while MDFA is conducted once for each B&B node (except at root node).

1	If root node of B&B tree, apply TSA-CG; else
2	If $iteration = 1$,
3	(MDFA): sort arcs in F_1 according to expression (12) if $F_1 \neq \varnothing$.
	If this is not possible, STOP; the problem is infeasible.
4	run GFA (to generate $\hat{F}_0$);
5	run GERA (to revise G_E to G_{EF}).
6	run OA on G_{EF}.
7	If $iteration > 1$,
8	if $\left\lvert B \setminus \hat{F}_0 \right\rvert \geq \lambda$, run OA on G_{EF}; else run ROA on G_{EF} with set $B \setminus \hat{F}_0$.

Fig. 17. TSA-CG/B&B.

8.2.3. Computational complexity of TSA-CG/B&B

Proposition 8.3. TSA-CG/B&B runs in $O\left(\left|\Re\right| m\gamma + \left|\Re\right| m\vartheta_{\max} + \Theta\right)$ time for the first-time solution ($iteration = 1$) and each subsequent solution ($iteration > 1$) requires $O(\left\|G_{EF}\right\|) = O(\Omega)$ time, where $\left\|G_{EF}\right\|$ denotes the number of arcs in G_{EF}.

Proof. In comparison to TSA-CG, TSA-CG/B&B adds MDFA operations in lines 3-5. Line 3, which sorts arcs in F_1 according to expression (12), can be done in $O(A) = O(m)$ time; line 4 implements GFA, which generates $\hat{F}_0$ and runs in $O(A) = O(m)$; and line 5 applies GERA, revising G_E to form G_{EF} and running in $O(A_E) = O(\Omega)$ because each arc in G_E is processed in constant time. Since the worst-case complexity of TSA-CG is $O\left(\left|\Re\right| m\gamma + \left|\Re\right| m\vartheta_{\max} + \Theta\right)$ (Proposition 8.2), the run time of TSA-CG/B&B is $O\left(\left|\Re\right| m\gamma + \left|\Re\right| m\vartheta_{\max} + \Theta\right) + O(m) + O(\Omega)$ $= O\left(\left|\Re\right| m\gamma + \left|\Re\right| m\vartheta_{\max} + \Theta\right)$. By Propositions 5.10 and 8.1, the second part is straightforward. ■

8.2.4. MDFA tests

This section describes tests of MDFA. We choose the traditional method for dealing with fixed arcs as a benchmark; it assigns large costs to forbidden arcs and small costs to prescribed arcs

and then applies OA with respect to these adjusted arc costs. This section studies instance *s*-1000-2a, using a factorial experiment design with four specified sets of F_1 and two specified sets of F_0, as shown at the bottom of Table 12.

Table 12 shows results related to all (F_1, F_0) combinations. Columns 5-8 give the specified sets of F_1 and F_0. Column 9 denotes the cardinality of $\hat{F}_0$. Columns 10 and 11 give the number of nodes in G_{EF} and ratio $|V_{EF}|/|V_E|$, respectively; columns 12 and 13 give the number of arcs in G_{EF} and ratio $|A_{EF}|/|A_E|$, respectively. Columns labeled (a), (b), (c) and (d) are specified at the bottom of Table 12. Stage 3 of TSA uses OA to solve SPP on G_{EF}. The last column in Table 12 gives the break even number of replications for which the run times for MDFA and the traditional method are the same. If the number of replications is larger than this value, the total run time for MDFA is less than that of the traditional method; otherwise, the traditional method is faster than MDFA. The break even is calculated using

$$breakeven = \frac{100 \times cpu(\text{MDFA})}{cpu(\text{traditional method}) - cpu(\text{stage 3\_of\_TSA})}. \tag{13}$$

MDFA consumes run time to identify and remove forbidden arcs from G_E, but this "set up" time is off set by reducing the size of G_E significantly (for most of the tests, the graph is reduced more than 40% and 50%; even 86% and 96% (test21 and test22), in terms of the number of nodes and arcs, respectively), reducing run time correspondingly. This set up time is incurred once in revising G_E but a time saving that results from the smaller graph accrues at each replication. For 100 replications, the total time for MDFA (revising G_E to G_{EF} plus solving SPP on G_{EF} for 100 replications) is less than that for the traditional method. *breakeven* ranges from 4 to 23 for this set of tests.

Another observation relates to $\hat{F}_0$. Table 12 shows that $\hat{F}_0$ can be calculated very quickly (0 seconds). It is worth mentioning the cardinality of $\hat{F}_0$ ($\left|\hat{F}_0\right|$) because a large $\left|\hat{F}_0\right|$ results in a small G_{EF}. Table 12 shows that $\left|\hat{F}_0\right|$ is large for all the tests ($\left|\hat{F}_0\right| \geq 4{,}050$) although $\left|F_1\right|$ and $\left|F_0\right|$ are either 1 or 4, respectively, small numbers in comparison to $|A| = 10{,}435$. When $F_1 = \text{set } 3$ and $F_0 = \text{set } 6$, $\left|\hat{F}_0\right| = 9{,}456$, that is, 91% of the arcs in G are forbidden ($|A| = 10{,}435$). Further, $\left|\hat{F}_0\right|$ is much larger when $F_1 = \text{set } 1 = \{(273,504)\}$ than when $F_1 = \text{set } 2 = \{(503,504)\}$ (6,870 versus 4,050 for $F_0 = \text{set } 5$; 6,872 versus 4,052 for $F_0 = \text{set } 6$), because the span of arcs $(273,504)$ and $(503,504)$ are 231 and 1, respectively; the former is much larger than the latter. Tests with $\left|F_1\right| = 4$ (i.e., $F_1 = \text{set } 3$ and $F_1 = \text{set } 4$) give similar results. These observations validate Remark 4, which notes that it is likely that many implied forbidden arcs can be identified if the spans of prescribed arcs are large. Next, consider tests using $F_1 = \text{set } 1 = \{(273,504)\}$ and $F_1 = \text{set } 4 = \{(382,503),(503,504),(504,512),(512,546)\}$. Although set 1 prescribes only one arc and set 4 prescribes four arcs, $\left|\hat{F}_0\right|$ is larger for set 1 than for set 4. Because the span of arc $(273,504)$ is $231 = 504 - 273$, larger than the total span of path 382-503-504-512-546, which is formed by the arcs in set 4, which is $164 = 546 - 382$. Note that the value of $\left|\hat{F}_0\right|$ is not as sensitive to the set of prescribed arcs as it is to the set of forbidden arcs.

Table 12. Results of instance *s*-1000-2a with fixed arcs

test	η	$\lvert V_E\rvert$	$\lvert A_E\rvert$	$\lvert F_1\rvert$	F_1 set	$\lvert F_0\rvert$	F_0 set	$\lvert \hat{F}_0 \rvert$	$\lvert V_{EF}\rvert$	$\lvert V_{EF}\rvert/\lvert V_E\rvert$	$\lvert A_{EF}\rvert$	$\lvert A_{EF}\rvert/\lvert A_E\rvert$	run time (sec.) (a)	(b)	(c)	total	(d)	*breakeven*
1	0.1	38268	378134	1	set 1	1	set 5	6870	13427	0.35	85310	0.23	0	0.156	0.765	0.921	2.399	10
2						4	set 6	6872	13427	0.35	85283	0.23	0	0.157	0.763	0.920	2.380	10
3					set 2	1	set 5	4050	29381	0.77	215113	0.57	0	0.141	1.510	1.651	2.370	17
4						4	set 6	4052	29381	0.77	215078	0.57	0	0.140	1.530	1.670	2.374	17
5				4	set 3	1	set 5	9495	9157	0.24	24590	0.07	0	0.109	0.347	0.456	2.381	6
6						4	set 6	9496	9157	0.24	24587	0.07	0	0.110	0.312	0.422	2.375	6
7					set 4	1	set 5	5504	20183	0.53	123131	0.33	0	0.141	0.935	1.076	2.411	10
8						4	set 6	5506	20183	0.53	123100	0.33	0	0.157	0.988	1.145	2.459	11
9	0.5	111871	1239197	1	set 1	1	set 5	6870	35048	0.31	317905	0.26	0	0.515	2.031	2.546	6.988	11
10						4	set 6	6872	35048	0.31	317770	0.26	0	0.500	2.045	2.545	6.985	11
11					set 2	1	set 5	4050	85662	0.77	703631	0.57	0	0.360	4.381	4.741	6.967	14
12						4	set 6	4052	85662	0.77	703382	0.57	0	0.375	4.374	4.749	6.987	15
13				4	set 3	1	set 5	9495	33280	0.30	106433	0.09	0	0.344	1.031	1.375	6.955	6
14						4	set 6	9496	33280	0.30	106428	0.09	0	0.344	1.028	1.372	6.969	6
15					set 4	1	set 5	5504	54910	0.49	432095	0.35	0	0.422	2.763	3.185	7.024	10
16						4	set 6	5506	54810	0.49	431939	0.35	0	0.422	2.720	3.142	6.966	10
17	0.9	21039	239205	1	set 1	1	set 5	6870	2947	0.14	9708	0.04	0	0.110	0.187	0.297	1.091	13
18						4	set 6	6872	2947	0.14	9706	0.04	0	0.110	0.141	0.251	1.063	12
19					set 2	1	set 5	4050	13905	0.66	118796	0.50	0	0.094	0.673	0.767	1.083	23
20						4	set 6	4052	13985	0.66	118778	0.50	0	0.094	0.658	0.752	1.075	23
21				4	set 3	1	set 5	9495	3504	0.17	10391	0.04	0	0.062	0.172	0.234	1.082	7
22						4	set 6	9496	3504	0.17	10388	0.04	0	0.047	0.153	0.200	1.064	6
23					set 4	1	set 5	5504	8442	0.40	49620	0.21	0	0.110	0.380	0.490	1.110	16
24						4	set 6	5506	8442	0.40	49618	0.21	0	0.109	0.344	0.453	1.061	16

set 1 = {(273,504)}; set 2 = {(503,504)}; set 3 = {(130,161), (369,382), (546,593), (830,865)}; set 4 = {(382,503), (503,504), (504,512), (512,546)}; set 5 = {(145,195)}; set 6 = {(145,195), (390,451), (582,707), (922,960)}.

(a) - run time for generating $\hat{F}_0$; (b) - run time for revising G_E; (c) - run time for solving SPP on G_{EF} for 100 replications;

(d) - run time of traditional method for 100 replications.

8.2.5. Summary and discussion

As shown in this section, TSA-CG/B&B incorporates MDFA to deal with fixed arcs; it detects infeasibility caused by fixed arcs in three steps: sorting arcs in F_1 detects infeasibility with respect to prescribed arcs (Corollary 8.1); GFA detects infeasibility due to G_R being disconnected; and GERA detects infeasibility due to G_E being disconnected.

MDFA is designed for application in which RCSP is solved repeatedly. It requires some computational time to remove all forbidden arcs from G_E, but each solution on the resulting graph G_{EF} can be found in much less time than on G_E because the run time required to solve SPP depends on the size of input graph and G_{EF} is smaller than G_E. The computational results show that if RCSP is solved repeatedly, MDFA outperforms the traditional method (Jaumard et al. (1996)), which is to assign a large cost to each forbidden arc and a small cost to each prescribed arc before solving SPP on G_E with respect to these adjusted arc costs.

Remark 5. Implied forbidden arcs must receive correct treatment in selecting branching variables. Consider a node in the B&B search tree with specified sets F_1, F_0, and generated $\hat{F}_0$. Decision variables corresponding to arcs in $\hat{F}_0$ should not be selected as branching variables, because, for each, the right child node (fixing the branching variable to 1) would be infeasible, and the left child node (fixing the branching variable to 0) would be the same as its parent node, which already forbids this arc (i.e., fixes the branching variable to 0). In the implementation of CG/B&B, $\hat{F}_0$ should be stored in association with each active node of the B&B tree to avoid such ineffectiveness. Note that the construction of $\hat{F}_0$ can be expedited by starting with the $\hat{F}_0$ constructed by GFA at its parent node.

Remark 6. In the implementation of a B&B approach, in which RCSP is a subproblem in CG/B&B, bottleneck arcs are forbidden at all nodes in the B&B tree. Thus, the variables corresponding to bottleneck arcs have zero values permanently and can be removed by fixing them to zero at root node.

CHAPTER IX

THREE EXTENSIONS OF TSA

This chapter presents extensions of TSA to solve SPPRW and a generalized resource-constrained SPP with both resource-limitation and resource-window constraints (SPRCRW) (Section 9.1); resource-constrained *k*-SPP (RC*k*SP) (Section 9.2); and multiple-resource, multiple-choice knapsack problem (MMCKP) (Section 9.3). Section 9.4 presents an application of MMCKP in a international assembly system design problem.

9.1. SPPRW and SPRCRW

TSA can be adapted to solve SPPRW, for which resource windows are given for each node but may not be tight. Thus, an adaptation of TSA (ATSA) for solving SPPRW uses stage 1 to tighten resources windows that are given initially. Desrochers et al. (1992) introduced a technique for tightening time windows (i.e., with only one type of resource) that has been widely used and proven to be effective in practice. We generalize this technique for SPPRW (i.e., to deal with multiple types of resource windows) and use it in stage 1 of ATSA. Figure 18 details our stage 1 algorithm in ATSA (S1A-A). At the end of S1A-A, it is likely that some arcs can be deleted. We refer the reader to Desrochers et al. (1992) for an illustration of their technique. Stages 2 and 3 of ATSA are exactly the same as stages 2 and 3 of TSA.

step 1. for each resource $r \in \Re$, for each $v_j \in V$ in increasing order of j,
apply the following four conditions to tighten resource windows:

(i) compute minimal resource requirement from predecessors:

$$\underline{t}_{jr} = \max\left\{\underline{t}_{jr}, \min\left\{\overline{t}_{jr}, \min_{(i,j)\in A}\left\{\underline{t}_{ir} + u_{ijr}\right\}\right\}\right\};$$

(ii) compute minimal resource requirement from successors:

$$\underline{t}_{jr} = \max\left\{\underline{t}_{jr}, \min\left\{\overline{t}_{jr}, \min_{(j,i)\in A}\left\{\underline{t}_{ir} - u_{jir}\right\}\right\}\right\};$$

(iii) compute maximal resource requirement from predecessors:

$$\overline{t}_{jr} = \min\left\{\overline{t}_{jr}, \max\left\{\underline{t}_{jr}, \max_{(i,j)\in A}\left\{\overline{t}_{ir} + u_{ijr}\right\}\right\}\right\};$$

(iv) compute maximal resource requirement from successors:

$$\overline{t}_{jr} = \min\left\{\overline{t}_{jr}, \max\left\{\underline{t}_{jr}, \max_{(j,i)\in A}\left\{\overline{t}_{ir} - u_{jir}\right\}\right\}\right\};$$

until no more reductions are possible.

Fig. 18. S1A-A: stage 1 algorithm of ATSA.

This approach can be generalized to SPRCRW (i.e., SPP with both resource-limitation constraints and resource-window constraints). Let $\Re_1$ be the set of resources limited by constraint (14b) and $\Re_2$ be the set of resources constrained by resource windows (14c)-(14d). Then, $\Re = \Re_1 \cup \Re_2$. Let $\overline{\Re} = \Re_1 \cap \Re_2$. Using the notation introduced in Chapter IV, a formal description of SPRCRW can be stated as MIP $(\wp 3)$:

$$(\wp 3) \qquad \min \quad z = \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{14a}$$

$$\text{s.t.} \quad \text{Constraints (1b) and (1d)}$$

$$\sum_{(i,j)\in A} u_{ijr} x_{ij} \le T_r \qquad \forall\, r \in \Re_1 \tag{14b}$$

$$\underline{t}_{jr} \le t_{jr} \le \overline{t}_{jr} \qquad j = 1,\ldots n,\ \forall\, r \in \Re_2 \tag{14c}$$

$$t_{ir} + u_{ijr} - t_{jr} \le M(1 - x_{ij}) \qquad \forall (i,j) \in A,\ \forall\, r \in \Re_2. \tag{14d}$$

In SPRCRW, resource $r \in \overline{\Re}$ involves both resource limitation (14b) and resource window constraints (14c)-(14d) if $\overline{\Re} \neq \varnothing$. If $\Re_2 = \varnothing$, $(\wp 3)$ reduces to $(\wp 1)$, RCSP; and if $\Re_1 = \varnothing$, $(\wp 3)$ reduces to $(\wp 2)$, SPPRW. A typical application of model $(\wp 3)$ is as the subproblem used in CG to solve VRPTW, in which multiple resource constraints represent vehicle capacity constraints (14b) and time-window constraints (14c)-(14d). A generalization of TSA (GTSA)

can solve SPRCRW. Figure 19 details our stage 1 algorithm for GTSA (S1A-G), which is designed specifically to solve SPRCRW.

step 1. For $r \in \Re_1$, apply S1A to formulate the window of resource r at each node ;
step 2. For $r \in \overline{\Re}$, form the union of the resource window that results from step 1 and the resource window that is given in model $(\wp 3)$ as the initial resource window for resource r at each node;
step 3. For $r \in \Re$, apply S1A-A to tighten the resource window for each node.

Fig. 19. S1A-G: stage 1 algorithm of GTSA.

In Figure 19, S1A-G transforms resource-limitation constraint (14b) to resource-window constraints (14c)-(14d) using S1A and then tightens the resource windows, if possible, using S1A-A. For $r \in \overline{\Re}$ (then $r \in \Re_1$), step 1 transforms constraint (14b) to a resource-window at each node, then step 2 takes the union of the resource window from step 1 and the resource window that is given initially in model $(\wp 3)$ for each node, and, finally, step 3 applies S1A-A to tighten the resulting resource windows. Stages 2 and 3 of GTSA are exactly the same as stages 2 and 3 of TSA.

TSA can be adapted easily to deal with RCSP, SPPRW and SPRCRW by adapting stage 1 appropriately. Stages 2 and 3 are applicable to each of these problems.

9.2. RC*k*SP

A *k*-SPP is to find the first *k* shortest paths. Efficient algorithms for *k*-SPPs were proposed by Yen (1971) for general graphs; Katoh et al. (1982) for undirected graphs; and Fox (1978), Eppstein (1998) and Lawler (1976) for acyclic paths. Eppstein (1998) gives a recent survey on *k*-SPP. In contract, RC*k*SP has not been studied. TSA can be adapted to solve RC*k*SP. Recall that the preliminary phase of TSA relaxes resource constraints by generating expanded graph G_E. The preliminary phase of TSA can transform RC*k*SP into an unconstrained *k*-SPP on the

expanded graph G_E. Thus, by incorporating a classical unconstrained *k*-SPP algorithm in stage 3 of TSA (e.g., Lawler's algorithm), TSA can solve RC*k*SP on an acyclic graph and is suitable for solving RC*k*SP repeatedly as in CG.

9.3. MMCKP

This section shows that TSA can solve MMCKP, in particular, when MMCKP is a subproblem in CG or CG/B&B. We illustrate this point after presenting a formal description of MMCKP.

Given κ sets $H_1,\ldots,H_\kappa$ of items to pack in a knapsack with multiple capacity constraints (i.e., resource limitations) $T_1,\ldots,T_{|\Re|}$ and that each item $j \in H_i$ has profit $-c_{ij}$ and resource requirement vector $\mathbf{u}_{ij} = \left\{u_{ij1},\ldots,u_{ij|\Re|}\right\}$, MMCKP is to choose exactly one item from each set so that the total profit is maximized without exceeding resource limitations T_r, $r \in \Re$. MMCKP may thus be formulated as model $(\wp 4)$:

$$(\wp 4) \qquad \max\ z = \sum_{i=1}^{\kappa}\sum_{j\in H_i} -c_{ij}x_{ij} \tag{15a}$$

$$\text{s.t.}\quad \sum_{i=1}^{\kappa}\sum_{j\in H_i} u_{ijr}x_{ij} \le T_r \qquad \forall r \in \Re \tag{15b}$$

$$\sum_{j\in H_i} x_{ij} = 1 \qquad i = 1,\ldots,\kappa \tag{15c}$$

$$x_{ij} \in \{0,1\} \qquad i = 1,\ldots,\kappa,\ j \in H_i\,. \tag{15d}$$

All coefficients u_{ijr} and T_r are positive, discrete values; coefficients c_{ij} are unrestricted; and sets $H_1,\ldots,H_\kappa$ are mutually disjoint with H_i having cardinality of h_i. The total number of items that are available to choose is $m = \sum_{i=1}^{\kappa} h_i$.

If $|\Re| = 1$, MMCKP reduces to the classical multiple-choice knapsack problem (MCKP), which is defined as a 0-1 knapsack problem (KP) with additional, disjoint multiple-choice constraints (15c). MCKP is NP-hard as it contains KP as a special case, but it can be solved in

pseudo-polynomial time using DP (Dudzinski and Walukiewicz (1987)). The problem has a wide range of applications: capital budgeting (Nauss (1978)), for which only one project may be selected from some subset of projects, subject to a single, scarce resource limitation; determining which components should be linked in series in order to maximize fault tolerance (Sinha and Zoltners (1979)); and menu planning (Sinha and Zoltners (1979)). MCKP often arises as a subproblem in CG (e.g., Wilhelm et al. (2005b) and Fisher (1981)). Algorithms available to solve MCKPs are typically based on B&B (Nauss (1978), Sinha and Zoltners (1979)) and DP (Dudzinski and Walukiewicz (1987) and Pisinger (1994)). Research on MMCKP is sparse. TSA gives a new and effective algorithmic approach to solve MMCKP (MCKP); in addition, it is suitable for solving MMCKP repeatedly as a subproblem in CG and CG/B&B.

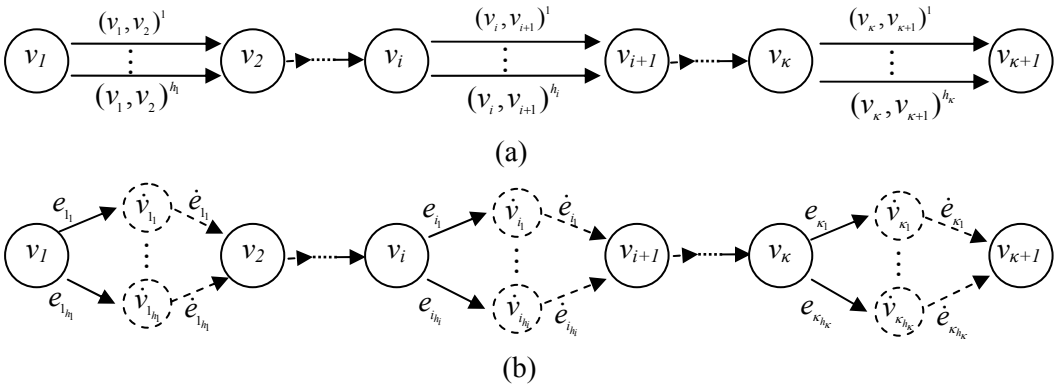


Fig. 20. Representation of MMCKP on an acyclic graph: (a) multigraph; (b) MMCKP-graph.

MMCKP can be represented as RCSP on an acyclic graph. Figure 20(a) uses an acyclic multigraph (with parallel arcs connecting certain pairs of nodes) to formulate MMCKP with parallel arc $(v_i, v_{i+1})^j$ representing item j in set H_i. Such a special multigraph has node set V with $|V| = \kappa + 1$; each arc has tail at v_i and head at v_{i+1}, $i = 1, \ldots, \kappa$. Arc $(v_i, v_{i+1})^j$ appears in the graph if and only if there is a corresponding decision variable x_{ij} in MMCKP. To avoid parallel

arcs, we construct an equivalent simple graph G (i.e., with no parallel arcs) by adding node $\dot{v}_{i_j}$ in the middle of arc $(v_i, v_{i+1})^j$, as shown in Figure 20(b). Such a node is called an arc-node and the set of all arc-nodes is denoted $\dot{V}$. The node set of graph G become $V + \dot{V}$ with $\left|V + \dot{V}\right| = m + \kappa + 1$. G comprises two types of arcs: type 1 arcs, denoted as e_{i_j}, point from node v_i to arc-node $\dot{v}_{i_j}$; type 2 arcs, denoted as $\dot{e}_{i_j}$, point from arc-node $\dot{v}_{i_j}$ to node v_{i+1} for $i = 1, \ldots, \kappa$. These two types of arcs appear in pairs. The arc set of G is denoted A with $|A| = 2m$. The resource requirement vector and the cost associated with e_{i_j} are $\mathbf{u}_{ij}$ and c_{ij}, and the resource requirement vector and the cost associated with $\dot{e}_{i_j}$ are the zero vector and zero, respectively. Such a special graph $G(V + \dot{V}, A)$ is called an MMCKP-graph because MMCKP as defined in model $(\wp 4)$ is equivalent to RCSP on this graph with the arc costs and resource requirement vectors as defined above. The equivalent RCSP is to find a shortest path from v_1 to $v_{\kappa+1}$ with respect to c_{ij} so that the total requirement of resource r observes its limit T_r for $r \in \Re$. The optimal value of MMCKP is the negative of the minimum cost for RCSP. The following propositions relate to the application of TSA to solve RCSP on the MMCKP-graph.

Proposition 9.1. Three conditions are equivalent: (a) arc e_{i_j} is bottleneck; (b) arc $\dot{e}_{i_j}$ is bottleneck; and (c) arc-node $\dot{v}_{i_j}$ in $\dot{V}$ is bottleneck.

Proof. If any of arcs e_{i_j}, $\dot{e}_{i_j}$, or arc-node $\dot{v}_{i_j}$ is a bottleneck, then the other two are not on any $v_1 - v_{\kappa+1}$ path. Thus, they all must be bottlenecks. ■

By Proposition 9.1, it is sufficient to check whether arc e_{i_j} is a bottleneck without checking arc $\dot{e}_{i_j}$ and arc-node $\dot{v}_{i_j}$. Once arc e_{i_j} is judged to be a bottleneck, Proposition 9.1

establishes that arc $\dot{e}_{i_j}$ and arc-node $\dot{v}_{i_j}$ are bottlenecks. Bottleneck arcs e_{i_j} and $\dot{e}_{i_j}$ and arc-node $\dot{v}_{i_j}$ can all be deleted from the graph.

Let H_i' be a subset of H_i that is formed by removing item j from H_i if arcs e_{i_j} and $\dot{e}_{i_j}$ are bottlenecks and have been deleted from the MMCKP-graph. Let $\underline{M}_r = \sum_{i=1}^{\kappa} \min_{j \in H_i'} \{u_{ijr}\}$ and $\overline{M}_r = \sum_{i=1}^{\kappa} \max_{j \in H_i'} \{u_{ijr}\}$ for $r \in \Re$. Define $\overline{\Delta}_r = \overline{M}_r - T_r$ and $\underline{\Delta}_r = T_r - \underline{M}_r$. Using the notation of Section 5.1, we have the following proposition.

Proposition 9.2. For the MMCKP-graph, $\underline{b}_{ir} = \overline{f}_{ir} - \overline{\Delta}_r$ and $\overline{b}_{ir} = \underline{f}_{ir} + \underline{\Delta}_r$ for $v_i \in V$, $r \in \Re$.

Proof. By induction in decreasing order of i. When $i = \kappa + 1$, $\overline{f}_{\kappa+1,r} = \overline{M}_r$; $\underline{f}_{\kappa+1,r} = \underline{M}_r$, so the statement is true. Suppose the statement is true for $i > k$ and consider the case for which $i = k$; then, (1) $\overline{f}_{k+1,r} = \overline{f}_{kr} + \max_{j \in H_k'} \{u_{kjr}\}$; (2) $\underline{f}_{k+1,r} = \underline{f}_{kr} + \min_{j \in H_k'} \{u_{kjr}\}$; (3) $\overline{b}_{kr} = \overline{b}_{k+1,r} - \min_{j \in H_k'} \{u_{kjr}\}$; and (4) $\underline{b}_{kr} = \underline{b}_{k+1,r} - \max_{j \in H_k'} \{u_{kjr}\}$. By (1) and (4), we have $\underline{b}_{k+1,r} = \overline{f}_{k+1,r} + (\underline{b}_{kr} - \overline{f}_{kr})$ and by induction $\underline{b}_{kr} - \overline{f}_{kr} = -\overline{\Delta}_r$. Thus, $\underline{b}_{k+1,r} = \overline{f}_{k+1,r} - \overline{\Delta}_r$. Similarly, by (2) and (3), we can prove $\overline{b}_{k+1,r} = \underline{f}_{k+1,r} + \underline{\Delta}_r$. This completes the proof. ■

Proposition 9.3. If $u_{ijr} > \min_{j \in H_i'} \{u_{ijr}\} + \underline{\Delta}_r$ for some $r \in \Re$, then arc e_{i_j} is a bottleneck.

Proof. By S1A, arc e_{i_j} is a bottleneck if $\underline{f}_{ir} + u_{ijr} > \overline{b}_{jr}$ for some $r \in \Re$. By Proposition 9.2, $\overline{b}_{jr} = \underline{f}_{jr} + \underline{\Delta}_r$; thus, arc e_{i_j} is a bottleneck if $u_{ijr} > \underline{f}_{jr} - \underline{f}_{ir} + \underline{\Delta}_r = \min_{j \in H_i'} \{u_{ijr}\} + \underline{\Delta}_r$. ■

Proposition 9.4. If $\overline{\Delta}_r \le 0$, the r^{th} knapsack capacity constraint is redundant; and if $\overline{\Delta}_r \le 0$, $\forall r \in \Re$, the optimal solution is $x_{ij_i} = 1$ for $j_i = \arg\min_{j \in H_i'} \{c_{ij}\}$, $\forall i = 1, \cdots, \kappa$.

Proof. $\overline{\Delta}_r = \overline{M}_r - T_r \le 0$ implies $\overline{M}_r \le T_r$. This proves first part. If all knapsack capacity constraints are redundant, then the optimal solution will be $x_{ij_i} = 1$ for $j_i = \arg\min_{j \in H_i'} \{c_{ij}\}$, $i = 1, \cdots, \kappa$, by the greedy argument. ■

Proposition 9.5. If any node in V of MMCKP-graph is a bottleneck, then MMCKP is infeasible relative to knapsack capacity constraints (15b).

Proof. If any node in V of the MMCKP-graph is a bottleneck, then there is no connected path from v_1 to v_{k+1}. ■

Corollary 9.1. For MCKP (i.e., $|\Re| = 1$), if $T_r \ge \underline{M}_r$, $r \in \Re$, then arc $e_{i_{j'}}$ with $j' = \arg\min_{j \in H_i} \{u_{ijr}\}$ can not be a bottleneck.

Proof. If arc $e_{i_{j'}}$ is a bottleneck, then node v_i is a bottleneck and, by Proposition 9.5, MCKP is infeasible with respect to resource limitations, contradicting the fact that $T_r \ge \underline{M}_r$, $r \in \Re$. ■

Based on Propositions 9.1-9.5, Figure 21 details a specialized S1A (S1A-M) for TSA for solving RCSP on the MMCKP-graph. S1A-M is specialized for the MMCKP-graph. Step 2(i) assures that the problem is feasible; otherwise, the algorithm stops at step 2. Step 2(i) also checks and removes redundant resource constraints using Proposition 9.4 and, if all resource constraints are judged to be redundant during the iterative process (steps 2 and 3), the greedy algorithm prescribes the optimal solution in step 3. Step 2(ii) uses Proposition 9.3 to detect arcs that are bottlenecks relative to each resource; according to Propositions 9.1 and 9.2, we do not need to test for bottleneck nodes. Step 3 iterates step 2 until no reduction is possible. After deleting bottleneck arcs, steps 4(i-ii) use Proposition 9.2 to further simplify the calculations of $\underline{b}_{ir}$ and $\underline{f}_{ir}$ for nodes in V. Step 4(iii) calculates $\underline{f}_{i_j,r}, \underline{b}_{i_j,r}, \overline{f}_{i_j,r}, \overline{b}_{i_j,r}, \underline{t}_{i_j,r}$ and $\overline{t}_{i_j,r}$ for nodes in $\dot{V}$ easily because the in- and out-degrees of each node in $\dot{V}$ are 1.

step 1. Initialize graph G_R by assigning $V_R = V + \dot{V}$ and $A_R = A$.

step 2. For $r = 1 \cdots |\Re|$,

(i) calculate $\overline{\Delta}_r$ and $\underline{\Delta}_r$. If $\underline{\Delta}_r < 0$ (i.e., $T_r < \underline{M}_r$), STOP; the problem is infeasible. If $\overline{\Delta}_r \le 0$, resource constraint r is redundant and removed from the model.

(ii) For each $v_i \in V \setminus \{v_{\kappa+1}\}$, if $u_{ijr} > \min_{j \in H_i'} \{u_{ijr}\} + \underline{\Delta}_r$, for $j \in H_i'$, delete e_{i_j}, $\dot{e}_{i_j}$, and $\dot{v}_{i_j}$.

step 3. If $\overline{\Delta}_r \le 0$ for all $r \in \Re$, the optimal solution is given by Proposition 9.4, STOP; otherwise, if anything was deleted from the graph in step 2, go back to step 2.

step 4. For $r = 1 \cdots |\Re|$,

set $\underline{f}_{1r} = 0$, $\overline{f}_{1r} = 0$, $\underline{b}_{\kappa+1,r} = T_r$, and $\overline{b}_{\kappa+1,r} = T_r$.

(i) For each $v_i \in V \setminus \{v_{\kappa+1}\}$ in decreasing v_i index, calculate $\overline{b}_{ir} = \overline{b}_{i+1,r} - \min_{j \in H_i'} \{u_{ijr}\}$.

(ii) For each $v_i \in V \setminus \{v_1\}$ in increasing v_i index, calculate $\overline{f}_{ir} = \overline{f}_{i-1,r} + \max_{j \in H_i'} \{u_{ijr}\}$,

$\underline{b}_{ir} = \overline{f}_{ir} - \overline{\Delta}_r$, $\underline{f}_{ir} = \overline{b}_{ir} - \underline{\Delta}_r$, $\underline{t}_{ir} = \max\{\underline{f}_{ir}, \underline{b}_{ir}\}$ and $\overline{t}_{ir} = \min\{\overline{f}_{ir}, \overline{b}_{ir}\}$.

(iii) For each $\dot{v}_{i_j} \in \dot{V}$, calculate $\overline{f}_{i_j,r} = \overline{f}_{ir} + u_{ijr}$, $\underline{f}_{i_j,r} = \underline{f}_{ir} + u_{ijr}$, $\underline{b}_{i_j,r} = \underline{b}_{i+1,r}$, $\overline{b}_{i_j,r} = \overline{b}_{i+1,r}$,

$\underline{t}_{i_j,r} = \max\{\underline{f}_{i_j,r}, \underline{b}_{i_j,r}\}$ and $\overline{t}_{i_j,r} = \min\{\overline{f}_{i_j,r}, \overline{b}_{i_j,r}\}$.

STOP.

Fig. 21. S1A-M: S1A specialized for the MMCKP-graph.

After stage 1, stages 2 and 3 of TSA can be applied. TSA is suitable for solving RCSP on an MMCKP-graph repeatedly; thus, it is suitable for solving MMCKP as in CG.

9.4. An application of MMCKP

In this section, we apply TSA to solve a MMCKP subproblem repeatedly in a branch-and-price (CG/B&B) approach, which is used to solve a real problem called NAFTAP, and demonstrate the performance of TSA by a numerical example. NAFTAP is to prescribe a strategic design of an assembly system and its supporting supply chain in the international business environment; it was formulated specifically to model the terms under the North American Free Trade Agreement (NAFTA). The strategic design problem is to prescribe a set of facilities, including their locations, technologies, and capacities, as well as strategic aspects of its supporting supply chain,

selecting suppliers; locating distribution centers; planning transportation modes; and allocating target levels (i.e., amounts) for production, assembly, and distribution (Wilhelm et al. (2005b)). The objective is to maximize after-tax profits. Our previous paper (Wilhelm et al. (2005b)) presents a comprehensive MIP that models this complicated international design problem. It deals with multiple time periods, multiple (end) products with bills-of-materials (BOMs) that have multiple echelons, and multiple countries but focuses on the relationship between the U.S. and Mexico that was established by NAFTA. It also investigates international business issues raised by NAFTA, such as border crossing, transfer price, exchange rate, local content rule, safe harbor rules, etc. We refer the reader to Wilhelm et al. (2005b) for a detailed description of NAFTAP and the model.

The NAFATP model in Wilhelm et al. (2005b) was designed for a CG approach. CG applies Dantzig-Wolfe decomposition (Dantzig and Wolfe (1960); Bazaraa et al. (1990)) to the linear relaxation of the model and exploits the block angular structure to obtain a subproblem of type 1 (SP1) for each end product $p_e \in P^E$ (P^E is a set of end products). SP1s associated with $p_e \in P^E$ are mutually independent of each other. SP1 associated with $p_e \in P^E$ ($SP_1(e)$) prescribes a system design for a single end product p_e, including production, assembly and the supply chain, by selecting a subset of alternative facilities. To facilitate presentation, we use the term "component" to indicate raw material, an in-process-part, end product, or an end product in the distribution subsystem. Correspondingly, we use the term "alternative facility" to indicate a unique location and a set of technologies and capacities for each supplier, manufacturing\subassembly facility, assembly facility, or distribution center. We use the term "process" to indicate outsourcing, production, assembly, or stocking operations. Thus, we represent supplier, production, assembly, and distribution decisions in a common way.

Let P_e be the set of components that are required to assembly and distribute end product $p_e \in P^E$ and F_p be the set of alternative facilities that can process component p. For an end product $p_e \in P^E$, $SP_1(e)$ involves only binary decision variables y_{pf}^e for $p \in P_e$ and $f \in F_p$. y_{pf}^e equals 1 if facility $f \in F_p$ is open for processing $p \in P_e$ in assembly of end product p_e; otherwise 0. With this notation, $SP_1(e)$ can be formulated as follows.

$$SP_1(e): \min \quad Z_e^{SP1} = \sum_{p \in P_e} \sum_{f \in F_p} \psi_{pf}^e y_{pf}^e \tag{16a}$$

$$\text{s.t.} \quad \sum_{f \in F_p} y_{pf}^e = 1 \qquad \forall \text{ raw material } p \in P_e \tag{16b}$$

$$-\sum_{f \in F_p} y_{pf}^e \le -1 \qquad \forall \text{ non raw-material } p \in P_e \tag{16c}$$

$$\sum_{p \in P_e} \sum_{f \in F_p} G_{pf}^O y_{pf}^e \le \bar{L}_e \tag{16d}$$

$$y_{pf}^e \in \{0,1\} \qquad \forall p \in P_e,\ f \in F_p. \tag{16e}$$

The objective function (16a) minimizes the total reduced cost associated with decision variables y_{pf}^e where ψ_{pf}^e denotes the reduced cost associated with y_{pf}^e. Equality (16b) assures that a solution prescribes exactly one supplier to provide the raw material required by local content rules under NAFTA. Inequality (16c) assures that a solution prescribes at least one facility (distribution center) to manufacture each component (store the end product), allowing facility flexibility (e.g., one component may be processed in several facilities). Inequality (16d) invokes a budget (resource) limitation, assuring that the total fixed cost associated with prescribing facilities for end product p_e does not exceed an investment budget of $\bar{L}_e$ dollars. Parameter G_{pf}^O represents the fixed cost of opening $f \in F_p$ to process $p \in P_e$. This budget limitation is appropriate because each p_e may be viewed as a profit center that serves a unique market segment. Finally, constraints (16e) give binary restrictions.

Note that if we can transform inequality (16c) to an equality, $SP_1(e)$ becomes MCKP as defined in Section 9.3 and can be solved as a RCSP on an MMCKP-graph using TSA. For this purpose, let S_p be a set of all nonempty subsets of F_p for $p \in P_e$ if p is not a raw material; and S_p be a set of singleton subsets of F_p if p represents a raw material. Then, $|S_p| = 2^{|F_p|} - 1$ if p is not a raw material and $|S_p| = |F_p|$ if p is a raw material. Define binary variable $\bar{y}^e_{ps}$ (for $p \in P_e$ and $s \in S_p$) that equals 1 if a set of facilities $s \in S_p$ is prescribed to process p; otherwise 0. Let $\bar{\psi}_{ps} = \sum_{f \in s} \psi_{ps}$ and $\bar{G}^O_{ps} = \sum_{f \in s} G^O_{pf}$ for $s \in S_p$. Then, $SP_1(e)$ can be reformulated as $\overline{SP_1}(e)$, which is MCKP and can be solved as RCSP on an MMCKP-graph using TSA.

$$\overline{SP_1}(e): \min \quad Z_e^{SP1} = \sum_{p \in P_e} \sum_{s \in S_p} \bar{\psi}^e_{ps} \bar{y}^e_{ps} \tag{17a}$$

$$\text{s.t.} \quad \sum_{s \in S_p} \bar{y}^e_{ps} = 1 \qquad \forall p \in P_e \tag{17b}$$

$$\sum_{p \in P_e} \sum_{s \in S_p} \bar{G}^O_{ps} \bar{y}^e_{ps} \le \bar{L}_e \tag{17c}$$

$$\bar{y}^e_{ps} \in \{0,1\} \qquad \forall p \in P_e,\ s \in S_p. \tag{17d}$$

Intuitively, we can construct the corresponding MMCKP-graph from the BOM network, which defines relationships amongst components that constitute end product p_e. In the BOM network, an arc connecting one node to another node means that the tail component is used to produce or assemble the arrowhead component. We offer an example (Figures 22) to help interpret the construction of the MMCKP-graph. The example deals with a single end product $p_e = p_5$, with the BOM network shown in Figure 22(a). The example assumes two alternative facilities for each component. In an actual application, there may be many alternative facilities.

Starting with a BOM network, we first construct a network – the BOM with alternative facilities (BAFN) – by adding nodes p_D and D after node p_e, and by adding parallel arcs with each arc representing a set of alternative facilities in S_p in which each tail component can be

processed. Figure 22(b) depicts BAFN and shows that, for example, component p_4 can be manufactured in alternative facilities f_5, f_6, or both. Nodes in level 1 always represent raw materials, which are, by assumption, outsourced, and corresponding arcs represent their respective, alternative suppliers. Node p_D represents end product p_e in the distribution subsystem, arcs from nodes p_e to p_D represent alternative final assembly facilities, node D represents all customer demands for the end product, and arcs from nodes p_D to D represent alternative DC facilities to store the end product p_e.

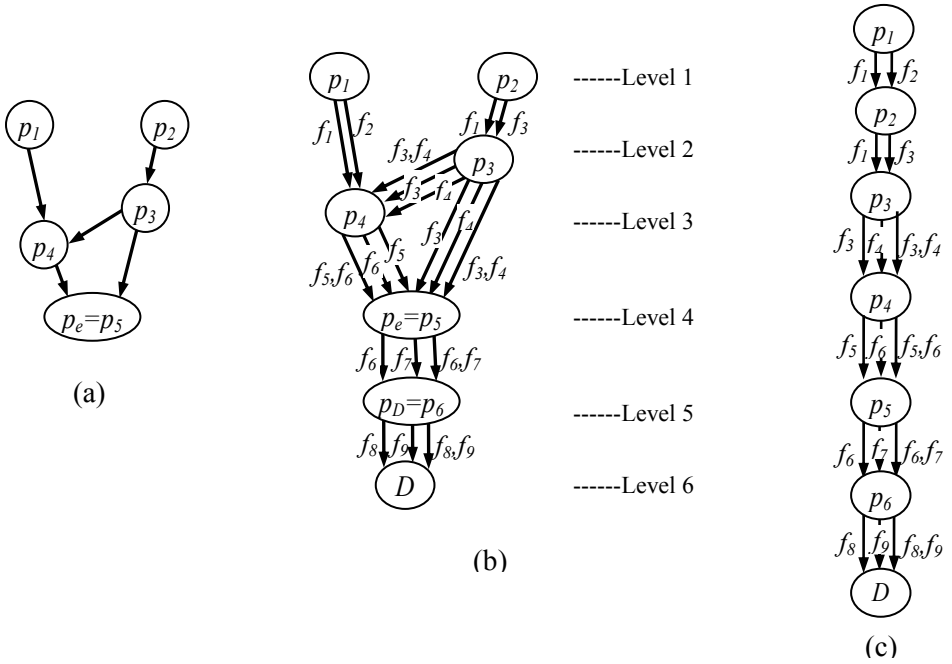


Fig. 22. An example of SP1: (a) example of BOM restrictions; (b) example of BAFN; and (c) example of SBAFN.

Then, we revise BAFN, forming a serial network (SBAFN) in which each node still represents a component and each arc represents a specified set of alternative facilities for processing the tail component, as shown in Figure 22(c). SBAFN includes one copy of every node p in BAFN. The nodes are sequenced according to the topological order in BAFN.

Actually, the nodes can be sequenced in any order. SBAFN is compatible with the multigraph in Figure 21(a). We can obtain the MMCKP-graph from SBAFN, by adding an arc-vertex in each arc in SBAFN. Associating the resource requirement $\bar{G}^{o}_{ps}$ and cost $\bar{\psi}_{ps}$ to the appropriate arc (that has the tail at node p and represents $s \in S_p$) as illustrated in Section 9.3, $\overline{SP_1}(e)$ (i.e., $SP_1(e)$) can be solved as RCSP on the specified MMCKP-graph using TSA. The optimal solution prescribes an optimal system design for end product p_e that observes budget limits.

We carried out a preliminary numerical example which involves two end products (e_1 and e_2) representing two types of laptop computers. These two end products have the same BOM and each end product has 12 raw materials (each with 3 alternative facilities), 3 in-process-parts (each with 4 alternative facilities), one end product (each with 4 alternatives facilities), and in addition, four alternative distribution centers, totally entailing 56 binary variables y^{e}_{pf}, equivalently, 111 binary variables $\bar{y}^{e}_{ps}$. The example has a planning horizon comprising three time periods and specifies cost parameters and demands randomly.

Branch-and-price solves this laptop example in 26.938 seconds, exploiting 27 B&B nodes. It calls TSA to solve $SP_1(e_1)$ and $SP_1(e_2)$ each 274 times and the total run time to obtain these 548 solutions is only 1.538 seconds. The total run time of the preliminary phases of the two SP1s is 0.016 second. The resulting expanded graph has 24(24) nodes and 58(58) arcs for each of these two SP1s. The performance of TSA in solving SP1s in this laptop example is quite satisfactory. More computational results will be provided in a later working paper. This example demonstrates that TSA can be used to effectively solve subproblems in CG/B&B.

CHAPTER X

INTRODUCTION TO SCHEDULING PROBLEMS WITH SDS

Most practical scheduling problems involve setup times (costs). In general, setup includes work required to prepare a machine (or process) to produce parts of a given type, including setting jigs and fixtures, adjusting tools, and provisioning material. Because of their prevalence in, and importance to, industry and because of the challenges they present to solution methodologies, scheduling problems that involve sequence-dependent setup (SDS) have attracted the interests of many researchers. Lot-sizing is intimately related to scheduling and a significant body of literature deals with integrating these issues (Haase (1994), Potts and van Wassenhove (1992), Drexl and Kimms (1997), Karimi et al. (2003)). Typical studies seek to prescribe the schedule as well as lot sizes to minimize the average setup cost over all jobs and holding cost over the entire schedule.

Specific objectives of this dissertation research on reviewing the scheduling problems that involve SDS are: (i) an overview with emphasis on recent results, (ii) an integrated view of lot-sizing and SDS scheduling, (iii) a perspective of this line of research, and (iv) fertile opportunities for future research.

The problem of prescribing a sequence, even for a single machine with SDS with makespan as the objective, is equivalent to the traveling salesman problem (TSP) and is, therefore, NP-hard (Pinedo (2002)). This difficulty has motivated a number of solution methods, including optimizing methods (B&B, branch-and-cut (B&C), DP, and MIP solvers), hybrids (methods that combine B&B, DP, or MIP solvers with a heuristic), and heuristics (meta-heuristics such as genetic algorithms (GA), simulated annealing (SA), tabu search (TS), and greedy randomized adaptive search procedure (GRASP); methods based on TSP algorithms,

greedy algorithm; decomposition; dispatching rules; simulation; list scheduling). Each approach has unique characteristics that suit it for specific problems.

Several earlier papers have reviewed research related to setup times (e.g., Allahverdi et al. (1999) and Yang and Liao (1999)). In particular, Allahverdi et al. (1999) cited nearly 200 references that deal with setup issues, but most were published before the 1990's. They categorized setup as sequence-independent or sequence-dependent as well as batch and non-batch. Batch setups involve times (or costs) that are typically much larger between batches (i.e., "major") than those between jobs within a batch (i.e., "minor"). Batches are also called families. They addressed traditional configurations (single machine, parallel machines, flow shops, and job shops) and emphasized that future research should focus on objectives related to due dates.

Other review papers have focused on specific machine configurations. Cheng et al. (2000) reviewed research on flow shop scheduling problems with setup times. They presented a complexity hierarchy and classified research into four categories that involved sequence independence and dependence relative to both job and family setup times (see also Monma and Potts (1989)). Kim and Bobrowski (1994) categorized early (before 1988) job shop scheduling research relative to the job arrival pattern and listed only four references that dealt with SDS.

Yet other papers have focused on combined lot-sizing and scheduling. Potts and van Wassenhove (1992) reviewed work that combined batching, lot-sizing and scheduling, stressing that, up to 1992, few studies had considered this important set of inter-related decisions. Drexl and Kimms (1997) summarized more recent work by presenting MIP formulations for different single- and multi-level lot-sizing and scheduling problems. But these formulations do not incorporate SDS.

Table 13. Three-field notation $\alpha \mid \beta \mid \gamma$

The α field specifies the machine configuration:	
1	single machine
Fm	*m*-machine flow shop
FFc	flexible flow shop with *c* stages in series, each with a set of identical machines in parallel
FJc	flexible job shop with *c* work centers, each with a set of identical machines in parallel
Jm	*m*-machine job shop in which each job has its own predetermined routing
Pm	*m* identical machines in parallel
Qm	*m* uniform machines in parallel, each operating at a different speed
Rm	*m* unrelated machines in parallel, each with a unique processing time for a job.
The β field specifies any processing restrictions and constraints that may be relevant:	
block	blocking can occur in a flow shop because buffers have limited capacities
brkdwn	breakdown or shutdown of machines
$bs_{ij}\ (bs_{ijk})$	sequence-dependent batch setup time (or cost) (on machine *k*)
$d_j (\overline{d}_j)$	jobs have due dates (deadlines)
$d\ (\overline{d})$	all jobs have a common due date (deadlines), $d_j = d\ (\overline{d}_j = \overline{d})$
M_j	not all *m* machines in parallel are capable of processing job *j*
nwt	jobs cannot wait between operations in a flow shop
prmp	jobs can be preempted
prec	precedence constraints relate jobs
prmu	a permutation sequence is used in a flow shop
r_j	jobs have known release dates
recrc	jobs may recirculate to be processed on the same machine several times
$s_{ij}\ (s_{ijk})$	sequence-dependent setup time (cost) for job *j* immediately after job *i* (on machine *k*)
The γ field describes the objective to be minimized:	
$C_{\max}$	makespan
$L_{\max}$	maximum lateness
$T_{\max}$	maximum tardiness
$\sum(w_j)C_j$	total (weighted) completion time
$\sum(w_j)T_j$	total (weighted) tardiness
$\sum(w_j)U_j$	(weighted) number of tardy jobs
$\sum E_j + \sum T_j$	total earliness / tardiness
$\sum w'E_j + \sum w''T_j$	total weighted earliness / tardiness
$\sum w'_j E_j + \sum w''_j T_j$	total weighted earliness / tardiness with unique penalties
$\sum s_{ij}, \sum s_{ijk}, \sum bs_{ij}$ or $\sum bs_{ijk}$	total setup time (cost) with respect to s_{ij}, s_{ijk}, bs_{ij}, or bs_{ijk}
$\Re$	any regular measure of performance (see Pinedo (2002))
γ	any measure of performance
Π	minimize cost
Π'	minimize sum (or average over time horizon) of setup and holding cost.

This dissertation contributes by focusing on recent results, providing a perspective of SDS scheduling research, integrating lot-sizing and SDS scheduling, and suggesting fertile opportunities for future research. Throughout, we use the three-field notation $\alpha|\beta|\gamma$ (Graham et al. (1979)) given in Table 13 to denote scheduling problems. For example, $1|s_{ij}|C_{\max}$ designates a single-machine configuration with SDS and the objective of minimizing makespan; $Fm|prmu,s_{ijk}|\sum w_jC_j$ designates an *m* machine flow shop configuration with SDS requiring a permutation sequence that minimizes total weighted completion time. We use standard terminology (e.g., Pinedo (2002)) in which a "sequence" is an ordering of jobs and a "schedule" is the set of starting and ending times for setup and production of every job. Symbols $\tilde{p}_j$, $\tilde{r}_j$ and $\tilde{s}_{ij}$ denote random processing, arrival and SDS, respectively.

This part of the dissertation has been organized in six chapters. Chapters X-XIV review studies related to the single machine, parallel machine, flow shop, and job shop configurations, respectively. We discuss research related to each configuration in detail, including solution methods, problem complexities, and combined lot-sizing and scheduling. Chapter XV presents our perspective, gives our conclusions, and suggests research opportunities.

CHAPTER XI

THE SINGLE MACHINE CONFIGURATION

Since the single machine represents a building block for more complex configurations, researchers have dealt with it in some detail. A fundamental issue is the inherent difficulty of single-machine scheduling problems that involve SDS. Research has established the complexity of most single machine problems; for example, Pinedo (2002) showed that $1|s_{ij}|C_{\max}$ is strongly NP-hard. For the case of batch SDS times, Monma and Potts (1989) showed that $1|bs_{ij}|C_{\max}$, $1|bs_{ij}|\sum w_j C_j$, $1|bs_{ij}|L_{\max}$, and $1|bs_{ij}|\sum U_j$ are polynomial solvable when the number of batches is fixed and that $1|bs_{ij}|C_{\max}$, $1|bs_{ij}|L_{\max}$, and $1|bs_{ij}|\sum U_j$ are NP-hard for an arbitrary number of batches; Ghosh (1994) showed that $1|bs_{ij}|\sum C_j$ and $1|bs_{ij}|\sum w_j C_j$ are strongly NP-hard for an arbitrary number of batches, even when the jobs within a batch have the same processing time and weight, and proposed a DP formulation; Chen (1997) proved that $1|bs_{ij}|\sum w'_j E_j + \sum w''_j T_j$ is NP-hard, even for the case with two batches of jobs and a common due date and weight for all jobs in each batch and for the case with large due dates that do not restrict the solution.

This section reviews the extensive research on single machine scheduling problems that involve SDS. Section 11.1 reviews optimizing and hybrid methods and Section 11.2 considers heuristics. Section 11.3 discusses the combined lot-sizing and scheduling problem.

11.1 Optimizing and hybrid methods

Even though complexity analysis is not encouraging, researchers have developed approaches, typically based on B&B, DP, or MIP solvers, to prescribe optimal solutions.

As an early work, Barnes and Vanston (1981) combined B&B with DP to solve $1|s_{ij}|\sum w_j C_j + \sum s_{ij}$. For $1|prec, s_{ij}|L_{\max}$, Uzsoy et al. (1991) proposed a B&B algorithm, for which run times increased rapidly for problems with more than fifteen operations, and Uzsoy et al. (1992) developed DP algorithms for $1|prec, s_{ij}|L_{\max}$ and $1|prec, s_{ij}|\sum U_j$ where precedence constraints comprise a number of strings (i.e., chains). Coleman (1992) formulated a MIP for $1|s_{ij}|\sum w' E_j + \sum w'' T_j$ and solved it to optimality using LINDO. Chen (1997) introduced DP algorithms for $1|bs_{ij}|\sum w'_j E_j + \sum w''_j T_j$ with an unrestrictively large, common due date for all jobs (i.e., in all batches), for $1|bs_{ij}|\sum w'_j E_j + \sum w''_j T_j + \sum \beta_j d_j$ with common due date for all jobs, where the term $\beta_j d_j$ represents a penalty assessed for the due date prescribed by decision variables d_j for each batch; and for $1|bs_{ij}|\sum w'_j E_j + \sum w''_j T_j + \sum \beta_j d_j$ with two batches of jobs. Rabadi et al. (2003) reported a B&B algorithm for $1|d, s_{ij}|\sum E_j + \sum T_j$. Asano and Ohta (1996) studied $1|\overline{d}_j, r_j, s_{ij}|\sum E_j$, detailing a B&B algorithm that included a dominance relationship to derive a strong lower bound. The branching rule and the strong lower bound enhanced computational effectiveness, but run time increased exponentially with the number of jobs (as to be expected), making it impractical to optimize large-scale instances (for example, a 30-job instance required about 30 minutes to solve). Asano and Ohta (1999) initiated a B&B algorithm to solve $1|brkdwn, r_j, s_{ij}|T_{\max}$ optimally for the case in which a set of break downs are pre-specified (e.g., representing preventative maintenance) and a post-processing procedure to delay a shutdown by prescribing its starting time with the goal of reducing $T_{\max}$. This work extended that of Leon and Wu (1992), which treated sequence-independent setup times. Tan et al. (2000) compared B&B (Ragatz (1993)) with three heuristics for $1|s_{ij}|\sum T_j$ (see also Section 11.2) and indicated that B&B may be preferred in solving smaller problems.

Only one paper has addressed rescheduling in single machine scheduling problems that involve SDS. Unal et al. (1997) rescheduled $1|bs_{ij},\overline{d}_j,r_j|\sum w_jC_j$ and $1|bs_{ij},\overline{d}_j,r_j|C_{\max}$ in a make-to-order environment with batch setup, inserting newly arrived jobs into a given sequence of existing jobs without making existing jobs tardy, without modifying the sequence of existing jobs relative to each other, and without incurring any additional setups. They related an exact, polynomial time algorithm for the $C_{\max}$ problem; and, after showing that the $\sum w_jC_j$ problem is strongly NP-hard, described two heuristics with data-dependent worst-case error bounds.

Several hybrid approaches have combined an optimizing method with a heuristic to resolve some portion of the problem. Ozgur and Brown (1995) suggested a two-phase hybrid for applications with a symmetric batch setup matrix $\{bs_{ij}\}$ (i.e., with $bs_{ij}=bs_{ji}$, $i,j=1,\ldots,n$); in application to an automated cable-assembly machine, their hybrid posted good results. Their first phase classifies products into families using cluster analysis and obtains an efficient sequence for each family by solving a TSP. Their second phase sequences families using a special-purpose B&B algorithm. Roslöf et al. (2002) combined a MIP solver with an iterative heuristic for solving $1|r_j,s_{ij}|w'C_{\max}+\sum w''T_j$.

11.2. Heuristics

The fact that SDS typically results in problems that are NP-hard has motivated many researchers to devise heuristics that prescribe "good" solutions within "reasonable" run times.

$1|s_{ij}|C_{\max}$ and $1|bs_{ij}|C_{\max}$ can be modeled as TSPs in which each node (city) represents a job and each directed arc gives the travel time (or distance) between the nodes it connects. Ozgur and Brown (1995) described a two-phase TSP heuristic (Section 11.1). Choi et al. (2003) considered a GA for applications with an asymmetric $\{s_{ij}\}$ matrix. Spina et al. (2003)

presented an approach that integrated Constraint Logic Programming and a GA in application to a process that produced sheets for catalytic converters. In their tests, their approach converged rapidly to a "near-optimal" solution but performed well only under certain conditions; in particular, it required pre-processing techniques to reduce the size of the solution space. In an application involving CC assembly on a single machine, Rossetti and Stanford (2003) used clustering methods to group CCs and a nearest–neighbor heuristic to sequence groups, requiring CCs in a group to be processed consecutively.

Numerous researchers have studied problems with objectives related to due-dates. For $1|s_{ij}|\sum T_j$, Alidaee et al. (2001) applied a generalized greedy algorithm and França et al. (2001) were able to prescribe good solutions using a *memetic algorithm* (see Moscato (1989, 1999)), a GA combined with a local improvement procedure. Tan et al. (2000) compared four methods (B&B (Ragatz (1993)), GA (Rubin and Ragatz (1995)), SA (Tan and Narasimhan (1997a)), and random-start pair wise interchange (Rubin and Ragatz (1995))) for $1|s_{ij}|\sum T_j$. Their experiments suggested that SA and random-start pair wise interchange can yield good solutions for large-scale instances and that B&B may be preferred in solving smaller problems (see also Section 11.1).

To solve $1|s_{ij}|\sum w_j T_j$, Kim et al. (1995) and Lee et al. (1997) used the *Apparent Tardiness Cost with Setups* (ATCS) heuristic (see Lee et al. (1992)). Kim et al. applied a feed-forward neural network to determine values of the look-ahead parameters in ATCS rules; their tests showed that, on average, their heuristic improved 19%-50% over Raman et al. (1989) and 9%-22% over Lee et al. (1992). Lee et al. (1997) used pre-determined parameters in ATCS rules and incorporated a local improvement procedure to further refine the schedule prescribed by the ATCS rule. Their experimental results showed that the ATCS dispatching rule is, on average, better than Raman's rule by more than 30% when the number of jobs is large (more than 40).

Ovacik and Uzsoy (1994b) developed heuristics for application on a rolling horizon basis to a case of $1|r_j,s_{ij}|L_{\max}$ that arose as a sub-problem in a job shop model of semiconductor production. At any time when a scheduling decision was to be made (e.g., when a machine completed service), they used B&B to solve $1|r_j,s_{ij}|L_{\max}$ for the set of jobs on hand as well as those with ready times within a specified horizon. To deal with dynamic job arrivals, they processed only the first job scheduled, then re-applied their method at the next decision point. Their tests showed that the rolling horizon approach can consistently give better schedules than dispatching rules in combination with local improvement procedures. Other heuristics include one based on Lagrangian relaxation for $1|r_j,s_{ij}|\sum w_jT_j^2$ (Sun et al. (1999)) and TS for $1|s_{ij}|\Pi$ (Laguna and Glover (1993), Laguna (1999), Kolahan et al. (1995), Kolahan and Liang (1998)) and $1|bs_{ij}|\Pi$ (Woodruff and Spearman (1992)), where Π denotes some setup related cost function (Table 14), and GRASP for $1|s_{ij}|\sum w_jC_j+\sum s_{ij}$ (Feo et al. (1996)) (Section 11.1, which discusses an early paper by Barnes and Vanston (1981)).

Relative to problems with multiple objectives, Tan and Narasimhan (1997b) used SA to minimize $\sum(w_j)T_j$ and $\sum s_{ij}$. Gupta and Sivakumar (2004) investigated a multi-objective problem in semiconductor manufacturing and employed simulation to minimize average cycle time and average tardiness while maximizing machine utilization. They introduced the concept of conjunctive simulated scheduling, in which discrete event simulation is used to evaluate scheduling criteria. When a machine becomes available, the simulator uses scheduling criteria to select the job for its next operation and then advances its clock to the next decision instance. They used compromise programming, which combines multiple objectives into one in making each decision.

11.3. Combined lot sizing and SDS scheduling

The problem of lot-sizing and scheduling on a single machine has received considerable attention. Six variations of this problem have been studied and are known to be NP-hard (Drexl and Kimms (1997)):

(1)economic lot scheduling (ELS) in which the planning horizon is infinite;

(2)capacitated lot-sizing (CLS), also called the large bucket model, identifies lots of several part types to be processed each period, then schedules jobs in each period separately;

(3)discrete lot-sizing and scheduling (DLS), also called the small bucket model, subdivides macro periods of CLS into micro periods in which only one part type may be processed at full capacity;

(4)continuous setup lot-sizing (CSL) adapts DLS, allowing at most one part type each period but using less than full capacity;

(5)proportional lot-sizing and scheduling (PLS) adapts CSL, allowing unused capacity to process a second part type in a period;

(6)general lot-sizing and scheduling GLS) incorporates a user-defined parameter to restrict the number of lots per period.

In particular, the case involving sequence-independent setup has been studied at some length (Thizy and van Wassenhove (1985), Dobson et al. (1987), Trigeiro et al. (1989), Fleischmann (1990), Cattrysse et al. (1993), Blocher et al. (1999)), for example, using B&B (Blocher et al. (1999)) and a heuristic based on column generation for a set-partitioning formulation (Cattrysse et al. (1993)). The case involving SDS has also attracted attention as discussed below.

Recently, Wagner and Davis (2002) developed a heuristic for ELS with SDS. Similar to the methods of Delporte and Thomas (1977) and Maxwell (1964), their heuristic imposes a cyclic schedule, which sequences all jobs in a cycle and then repeats the cycle indefinitely. They evaluated sequences using a nonlinear program and showed that their heuristic can outperform Dobson's heuristic (1992) when utilization is high and SDS time (cost) is significant. Dobson's

heuristic is based on a Lagrangian relaxation that leads to two types of sub-problems: one prescribes lot size; and the other, sequences by solving a TSP.

Haase and Kimms (2000) and Gupta and Magnusson (2005) studied CLS with SDS in which setups may be carried over from one period to the next and are preserved over idle periods. Haase and Kimms formulated a MIP that considered only efficient sequences. (An efficient sequence is one that no other sequence dominates. Sequence A is said to dominate sequence B if the total setup cost of A is less than that of B, A and B comprise exactly the same set of jobs, and the first and last jobs of the two sequences are the same.) They indicated that the size of solvable instances ranged from 3 products and 15 periods to 10 products and 3 periods and solved instances within this range optimally using B&B, incorporating a tailor-made enumeration method. Gupta and Magnusson (2005) also formulated a MIP and recounted a heuristic that solved test instances to within 10%-16% of optimum, depending on the size of the instance. Earlier work by Haase (1996) devised a heuristic priority rule for the case with SDS costs (but zero times) and used a local search to derive appropriate values of parameters for use by the priority rule. Miller et al. (1999) formulated a MIP that allowed backlogs (i.e., backorders) and introduced a GA that incorporated a hill-climbing technique to solve it.

Some research has addressed DLS with SDS. These DLS-SDS models subdivide the planning horizon into many short time periods (e.g. shifts or days), and require the processing of each lot to take several full time periods - the so-called “all-or-nothing” assumption. Fleischmann (1994) formulated this problem as a TSP with time windows and devised a procedure to determine lower bounds using Lagrangean relaxation in combination with a heuristic. He reported computational results for problems with up to 10 products and 150 periods. Salomon et al. (1997) developed an exact solution method by applying a DP algorithm to the problem formulated as a TSP with time windows and was able to optimize instances of

moderate size. Jordan and Drexl (1998) offered a B&B algorithm, employing specialized bounding and dominance rules.

Meyr (2000) extended the GLS of Fleischmann and Meyr (1997) to deal with SDS times. He formulated a MIP and proposed an approach that combined a dual algorithm to re-optimize sub-problems with a local search heuristic. He applied this approach to GLS with SDS by embedding a dual network flow algorithm into threshold accepting and SA, respectively. He used local search procedures (threshold accepting and SA) to fix the setup sequence and solved a network flow sub-problem for each candidate setup sequence to determine the lot sizes and holding costs associated with each candidate. Network flow sub-problems can be re-optimized quickly by using information about the current solution to evaluate a new candidate if these two solutions differ only slightly. He also applied this approach to the corresponding, parallel machine problem (Meyr (2002)) (Section 12.3).

CHAPTER XII

THE PARALLEL MACHINE CONFIGURATION

Production systems employ various types of parallel machine configurations and a substantial literature has focused on identifying the complexity of related problems (Monma and Potts (1989), Cheng and Chen (1994), Ghosh (1994)). This section reviews the research related to parallel-machine scheduling problems that involve SDS. Section 12.1 reviews optimizing and hybrid methods and Section 12.2 addresses heuristics. Section 12.3 discusses the combined lot-sizing and scheduling problem.

12.1. Optimizing and hybrid methods

Regrettably, little research has been directed to developing optimizing methods for the parallel machine configuration. Balakrishnan et al. (1999) formulated $Qm \mid r_j, s_{ijk} \mid \sum w'_j E_j + \sum w''_j T_j$ as a MIP with substantially fewer zero-one variables than required by typical formulations. They successfully applied their model to solve small instances and suggested use of Bender's decomposition to solve larger instances by separating their formulation into an integer master problem, which prescribes job assignments to machines and the sequence at each machine, and a linear programming sub-problem, which prescribes the exact completion time of each job.

One hybrid addressed a problem in the chemical industry that involved an order of magnitude difference between small and large setup times. Bitran and Gilbert (1990) formed a network to represent setup cost, a B&B algorithm to deal with the large setup times, and a heuristic to sequence within families. In another hybrid, Yalaoui and Chu (2003) described a heuristic for $Pm \mid s_{ij} \mid C_{\max}$ with job-splitting (i.e., each job can be split into segments that can be processed in parallel on different machines). They first decomposed the problem into

independent $1|s_{ij}|C_{\max}$ problems, which they modeled as TSPs and solved by the B&B algorithm of Little et al. (1963). Subsequently, they employed a step-by-step improvement methodology, taking setup times and job-splitting into account.

12.2. Heuristics

Most of the research on the parallel machine configuration has focused on heuristics. Guinet (1993) formulated a MIP for $Pm|s_{ij}|C_{\max}$ (and another for $Pm|s_{ij}|\sum C_j$)) to show that it is a vehicle routing problem and, thus, NP-hard. He proposed a heuristic based on the assignment-problem algorithm. For $Pm|s_{ij}|C_{\max}$, França et al. (1996) proposed a three-phase heuristic based on TS. Gendreau et al. (2001) devised lower bounds and a "divide and merge" heuristic that proved to be much faster than the heuristic of França et al. (1996) while providing solutions of similar quality. Mendes et al. (2002) compared two meta-heuristics for $Pm|s_{ij}|C_{\max}$: a heuristic based on TS (adapted from França et al. (1996)) and a memetic, which combines a GA with local search procedures. Results showed that the memetic was superior when setup times are small compared to processing times but TS excelled on instances with large setup times and many machines (6 or 8 machines in their tests).

Other research has addressed variations of $Pm|s_{ij}|C_{\max}$. For example, Kurz and Askin (2001) presented a MIP for $Pm|r_j,s_{ij}|C_{\max}$ and applied four heuristics based on earlier work (Papadimitriou and Steiglitz (1998), Johnson and Papadimitriou (1985), Reinelt (1994), and Coffman et al. (1978)). Weng et al. (2001) described and tested seven heuristics for $Rm|s_{ij}|\sum w_jC_j$.

Researchers have developed various heuristics to solve (weighted) tardiness problems: $Pm \mid s_{ij} \mid \sum(w_j)T_j$, $Qm \mid s_{ijk} \mid \sum(w_j)T_j$, and $Rm \mid s_{ijk} \mid \sum(w_j)T_j$. Following Lee et al. (1997), Lee and Pinedo (1997) offered a three-phase heuristic for $Pm \mid s_{ij} \mid \sum w_j T_j$; it comprises three phases: (1) a pre-processing procedure to calculate due-date related factors; (2) an ATCS dispatching rule to construct seed sequences; and (3) a SA heuristic that starts from a seed solution generated by the second phase. Based on Lee et al. (1992), Lee et al. (1997), and Kim et al. (1995), Park et al. (2000) extended the ATCS rule for $Pm \mid s_{ij} \mid \sum w_j T_j$, utilizing a neural network to determine values of the look-ahead parameters employed by the rule. Using a simulation model, they showed that their approach improved the objective by an average of 6% over the ATCS rule of Lee et al. (1997). Kim et al. (2003) adapted a TS heuristic for $Pm \mid s_{ij} \mid \sum w_j T_j$, categorizing jobs in accordance with due-dates. Their computational testing showed that, in general, the performance of their heuristic improved as the number of machines increased and dis-improved as the number of jobs increased. Bilge et al. (2004) applied TS to $Qm \mid r_j, s_{ijk} \mid \sum T_j$ and Kim et al. (2002) used SA for $Rm \mid d, s_{ijk} \mid \sum T_j$. The computational tests of Kim et al. (2002) showed that their heuristic achieved significantly better total tardiness values than a neighborhood search did.

Researchers have also studied the total (weighted) earliness and tardiness problem. Heady and Zhu (1998) solved $Pm \mid s_{ij} \mid \sum w'_j E_j + \sum w''_j T_j$ using a greedy heuristic, and Radhakrishnan and Ventura (2000) employed SA to solve $Pm \mid s_{ij} \mid \sum E_j + \sum T_j$. Both Balakrishnan et al. (1999) (see Section 12.1) and Sivrikaya-Serifoglu and Ulusoy (1999) investigated $Qm \mid r_j, s_{ijk} \mid \sum w' E_j + \sum w'' T_j$. Sivrikaya-Serifoglu and Ulusoy (1999) assumed $w' \neq w''$ and composed two GAs, one with a crossover operator and one without. Tests on 960

randomly generated instances indicated that their GAs were effective; that a neighborhood-exchange search gave relatively better results in small and easy instances; that GA with the crossover operator was best in application to larger, more difficult instances; and that the recombinative power of GA with the crossover operator improved with increasing problem size.

For $Pm|r_j,s_{ij}|L_{\max}$, Ovacik and Uzsoy (1995) extended the rolling-horizon heuristic they initiated for $1|r_j,s_{ij}|L_{\max}$ (Ovacik and Uzsoy (1994b) (Section 11.2)). Their tests showed that their heuristic outperformed the EDD dispatching rule combined with local search methods. Kim and Shin (2003) solved $Rm|r_j,s_{ij}|L_{\max}$ using a TS that restricts changes in job sequence, allowing only the jobs that can change the position of the job that defines $L_{\max}$ in the current schedule to change sequence position. Their tests showed that their restricted TS prescribed better solutions more quickly than the rolling horizon procedure of Ovacik and Uzsoy (1995) for $Pm|r_j,s_{ij}|L_{\max}$, and outperformed basic TS (Glover (1989, 1990)) and SA (Kirkpatrick et al. (1983)) for $Rm|r_j,s_{ij}|L_{\max}$.

List-scheduling algorithms are one-pass heuristics that are widely used to prescribe schedules. The "standard" list-scheduling algorithm constructs a schedule by assigning each job in listed order to the machine that becomes idle first. Ovacik and Uzsoy (1993) showed that, for $Pm|s_{ij}|C_{\max}$ and $Pm|s_{ij}|L_{\max}$, list schedules need *not* be dominant (A set of schedules is called *dominant* if it contains at least one optimal schedule). They also derived worst-case error bounds for the standard list-scheduling algorithm applied to $Pm|s_{ij}|C_{\max}$ and $Pm|s_{ij}|L_{\max}$, assuming that setup times are bounded by processing times. Schutten (1996) studied a list-scheduling algorithm for $Pm|r_j,s_{ij}|\Re$ in which each job is assigned in listed order to the machine that can finish it at the earliest time (i.e., start its processing (not its setup) as early as

possible). He proved that this algorithm yields dominant list schedules for $Pm|r_j, s_{ij}|\Re$. In contrast, Hurink and Knust (2001) showed that no list-scheduling algorithm can efficiently prescribe dominant schedules for $Pm|prec, s_{ij}|C_{\max}$.

Dhaenens-Flipo (2001) investigated $Qm|\bar{d}, s_{ijk}|\Pi$ in which Π is the sum of production, distribution, and setup costs. They relaxed deadline constraints by incorporating them in the objective function, which became a linear combination of two criteria, deadlines and cost, and then solved their model with a heuristic. In an application involving CC assembly on non-identical parallel machines, Hop and Nagarur (2004) proposed a composite GA to solve weighted, multiple objectives by dealing with workload balancing, CC similarities, and total setup time.

Several studies have addressed environments in which uncertainty must be considered explicitly; Aytug et al. (2005) reviewed this work. Arzi and Raviv (1998) modeled a workstation as $Rm|\tilde{r}_j, \tilde{s}_{ij}|\gamma$. They suggested several dispatching rules that proved, through simulation tests, to give good average values of several objective functions designated by γ: though put, total setup time, and work-in-process (WIP) level. Anglani et al. (2005) studied $Pm|\tilde{p}_j, s_{ij}|\sum s_{ij}$ in which SDS involved costs (but zero times). They formulated a fuzzy mathematical programming model and solved an approximate version of that model to minimize $\sum s_{ij}$.

12.3. Combined lot sizing and SDS scheduling

Several studies have addressed combined lot sizing and SDS scheduling on parallel machines. Kang et al. (1999) investigated $Rm|s_{ij}|\Pi$, which holds the objective, Π, of minimizing the sum of setup and holding costs minus sales revenue. They assumed that the demand for each

product must be satisfied in each time period but sales could exceed demands and that SDS involved costs but zero times. Their hybrid column generation approach implements B&B in a heuristic manner, chaining sub-sequences for a time period, then time periods for the scheduling horizon to form a sequence. One heuristic truncates the B&B tree based on the number of fractional variables in the optimal solution at a node; the other heuristic iteratively executes local search to find improvements in the neighborhood of an incumbent solution. Their computational tests showed that their heuristics required lengthy run times but prescribed good solutions with machine utilizations of 70% and 95% but poor solutions at higher utilizations (e.g., 99%). Meyr (2002) generalized the problem, allowing non-zero SDS times. This study extended Meyr's earlier work (Meyr (2000)), describing heuristics that combined dual re-optimization with either SA or threshold accepting. Meyr showed that his two heuristics prescribed solutions that were competitive with those prescribed by the approach of Kang et al. (1999) but that they required very long run times.

CHAPTER XIII

THE FLOW SHOP CONFIGURATION

The flow shop configuration is common in many manufacturing and assembly facilities; it comprises a series of machines that process on each job as it progresses down the line. For example, the typical CC assembly line can be modeled as a flow shop or flexible flow shop with SDS (Kurz and Askin (2003, 2004)).

An optimal permutation sequence, which uses one of the possible $n!$ permutations of jobs to order them at all machines, is typically sought. The alternative would be to prescribe a different sequence at each machine, but this would require searching over $(n!)^m$ sequences to prescribe an optimal solution, a truly daunting task. Unless otherwise noted, all flow shop-scheduling studies we review deal with permutation schedules.

Gupta (1986) proved that $Fm \mid s_{ijk} \mid \gamma$ is NP-Hard. Gupta and Darrow (1986) proved that $F2 \mid s_{ijk} \mid C_{\max}$ is NP-hard, and that permutation schedules for this problem do not always minimize $C_{\max}$.

This section reviews the rather copious research on flow shop scheduling that involves SDS. Section 13.1 addresses optimizing and hybrid methods and Section 13.2 reviews heuristics. Section 13.3 discusses several variations of the flow shop with SDS. Section 13.4 reviews the scheduling of CC assembly, an important industrial application of the flow shop with SDS. Finally, Section 13.5 discusses the combined lot-sizing and scheduling problem.

13.1. Optimizing and hybrid methods

A substantial literature has focused on versions of $Fm \mid s_{ijk} \mid C_{\max}$, especially using permutation schedules $Fm \mid prmu, s_{ijk} \mid C_{\max}$. For $Fm \mid prmu, s_{ijk} \mid C_{\max}$ and $Fm \mid nwt, s_{ijk} \mid C_{\max}$, Stafford and Tseng (2002) presented two MIP models, designated WST and SGST. WST utilizes n^2 (where n is the number of jobs) binary variables to assign jobs to sequence positions in a manner analogous to the classical assignment problem, and SGST uses pairs of disjunctive constraints and $n(n-1)/2$ binary variables to prescribe which job in each pair of jobs is sequenced ahead of the other. They compared these two models using a 12-cell experimental design and concluded that the models were competitive in solving instances with $n \leq 7$, and that the WST model proved significantly better for instances with $n \geq 8$.

Rios-Mercado and Bard (1998a) studied the convex hull of the set of feasible solutions for both a TSP-like MIP (in which binary variables prescribe whether one job is an immediate predecessor of another or not) and the formulation of Srikar and Ghosh (1986) (in which binary variables prescribe whether one job is a predecessor of another or not) (Stafford and Tseng (1990) reported a minor error in constraint formulation). They developed several classes of valid inequalities, showed that some of the inequalities are indeed facet-defining for the two different formulations, and implemented a B&C approach to test the effectiveness of the valid inequalities.

Rios-Mercado and Bard (1999a) devised a hybrid B&B algorithm with a partial enumeration strategy to find an approximate (or, with luck, an optimal) solution for $Fm \mid prmu, s_{ijk} \mid C_{\max}$. They derived a lower bound based on machine completion times in partial schedules and their tests showed this bound was more effective than that given by the linear relaxation of their model. To further improve their algorithm, they applied dominance rules in

B&B and used GRASP (Rios-Mercado and Bard (1998b)) and a TSP-based heuristic (Rios-Mercado and Bard (1999b)) to find good feasible solutions that provide upper bounds.

Relative to objectives based on due dates ($FFc \mid M_j, s_{ijk} \mid \sum E_j + \sum T_j$ and $FFc \mid M_j, r_j, s_{ijk} \mid \sum E_j$), Hui et al. (2000) and Mendez et al. (2001), respectively, gave MIP formulations for the case in which each stage contains non-identical parallel machines.

13.2. Heuristics

Recently, Ruiz et al. (2005) presented an advanced GA and a hybrid GA for $Fm \mid prmu, s_{ijk} \mid C_{\max}$ that applied a local search in an improvement phase. As a basis of comparison to evaluate the performance of their two GAs, they adapted the five flow shop heuristics that are considered the most capable in solving $Fm \mid prmu \mid C_{\max}$: SA (Osman and Potts (1989)), TS (Widmer and Hertz (1989)), GA (Reeves (1995)), iterated local search (Stutzle (1998)), and a GA originally proposed for the no-wait flow shop (Aldowaisan and Allahverdi (2003)). They tested four sets of 120 instances each that were based on the instances studied by Taillard (1993). These instances ranged from 20 jobs and 5 machines to 500 jobs and 20 machines and included SDS times published by Vallada et al. (2003). Test results showed that their two GAs outperformed the other heuristics and they concluded that their two GAs are the most effective methods available for solving $Fm \mid prmu, s_{ijk} \mid C_{\max}$. Simons (1992), Das et al. (1995), and Rios-Mercado and Bard (1998b, 1999b) have described other heuristics for $Fm \mid prmu, s_{ijk} \mid C_{\max}$. Additionally, Norman (1999) devised a TS heuristic for $Fm \mid block, prmu, s_{ijk} \mid C_{\max}$ with finite buffers. Bianco et al. (1999) formulated $Fm \mid nwt, r_j, s_{ijk} \mid C_{\max}$ as an asymmetric TSP with time window constraints and developed two greedy heuristics to solve it.

Several researchers have studied objectives based on due dates. Parthasarathy and Rajendran (1997a, 1997b) introduced SA for $Fm \mid prmu, s_{ijk} \mid \max(w_j T_j)$ and $Fm \mid prmu, s_{ijk} \mid (\sum w_j T_j)/n$. Extending their earlier work (Rajendran and Ziegler (1997)), Rajendran and Ziegler (2003) recently related a heuristic for $Fm \mid prmu, s_{ijk} \mid \sum w_j C_j + \sum w_j' T_j$ that constructs a "good" sequence using two preference relations and then applies an improvement scheme. In their tests, their heuristic was faster and prescribed solutions of higher quality than either a random search or a greedy local search.

13.3. Variations of the flow shop

Researchers have devised heuristics to address a number of variations of the flow shop. One variation of the traditional flow shop allows certain jobs to skip some stages. Kurz and Askin (2003, 2004) investigated $FFc \mid s_{ijk} \mid C_{\max}$ with this variation, proposing a MIP and comparing four heuristics: a naive greedy approach, a multiple machine-insertion TSP heuristic, Johnson's Rule, and a GA with random-keys. In their tests, the GA with random-keys was very effective. Pugazhendhi et al. (2004) investigated $Fm \mid s_{ijk} \mid C_{\max}$ and $Fm \mid s_{ijk} \mid \sum w_j C_j$, modifying a set of recursive equations to account for skipped operations in calculating the timetable for permutation schedules and developing a simple heuristic to derive non-permutation schedules from a given permutation sequence. Tests showed that the resulting non-permutation schedules consistently gave better $C_{\max}$ values than the associated permutation schedules. However, non-permutation schedules did not make significant improvements on average in minimizing $\sum w_j C_j$.

Another variation allows the reentry of jobs, which occurs, for example, in the production of integrated circuits. Pearn et al. (2004) presented three fast network algorithms for

$FFc \mid bs_{ijk}, \bar{d}_j, recrc \mid \sum bs_{ijk}$ based on a case study of integrated-circuit final testing. Hwang and Sun (1997, 1998) addressed a version of $F2 \mid recrc, s_{ij1} \mid C_{\max}$ in which SDS times on the first machine depend, not on the immediately preceding job, but on the job that is two sequence positions ahead of it and suggested a hybrid that combines DP and a GA. For $FFc \mid recrc, s_{ijk} \mid L_{\max}$, Demirkol and Uzsoy (2000) studied decomposition methods (Ovacik and Uzsoy 1997)), and Ovacik and Uzsoy (1994a) investigated dispatching rules that consider jobs available at the machine as well as others that will become available within a certain future time window.

Extending their previous work (Hwang and Sun (1997, 1998)), Sun and Hwang (2001) investigated a variation of $F2 \mid s_{ij2} \mid C_{\max}$ in which only the second machine has SDS times and they depend, not on the immediately preceding job, but on the job that is k sequence positions ahead. This version was motivated by a case study that involved machining a cylinder head. They solved this problem optimality using a DP algorithm, which utilized a dominance condition, and also proposed a GA heuristic.

13.4. CC assembly

In CC assembly, setup is time-consuming (Rossetti and Stanford (2003), Maimon et al. (1993), Hashiba and Chang (1991)) and sequence-dependent because different types of CCs share different subsets of component types. Thus, SDS must be considered explicitly in scheduling CC assembly. McGinnis et al. (1992) provided an overview of the essential elements of CC-assembly technologies as well as a framework for process-planning. Some applications involve CC assembly on a single machine (e.g., Rossetti and Stanford (2003)) or parallel machines (e.g., Hop and Nagarur (2004)). Rossetti and Stanford (2003) enumerated a method to estimate SDS

time based, not only on an existing setup, but also on all preceding setups. They identified component placement as a bottleneck in the CC assembly line and used a single machine to model all placement operations. Based on estimated SDS times, they used clustering methods to group CCs and a nearest–neighbor heuristic to sequence groups. Hop and Nagarur (2004) modeled a CC assembly process as a set of non-identical parallel machines and discussed a composite GA to solve it.

CC assembly has been well studied (e.g., Feo et al. (1995), Shailendra et al. (1996), Rajkumar and Narendran (1998), and Wilhelm and Tarmy (2003)). The typical CC assembly line can be modeled as a flow shop (e.g., Maimon et al. (1993), Kim et al. (1996), Logendran et al. (2003), and Schaller et al. (2000)) in which SDS setup involves provisioning each component type at the machine that places it on the CC. Some research on flow shop with SDS can be applied directly to schedule CC assembly (e.g., Kurz and Askin (2003, 2004)).

Maimon et al. (1993) modeled a CC assembly process as a two-machine flow shop and devised two scheduling methods that were based on component commonality among CC types. Their first method, GUB, sets up component types that are common among two or more CC types only once and assembles them onto their respective CC types; it then sequentially sets up and assembles remaining component types for each CC type. Their second method, SDSM, schedules CC types so that each is followed (immediately) in the schedule by the CC type with which it shares the largest number of component types. They gave a numerical example, which showed that GUB resulted in a higher throughput but also a higher WIP level than SDSM. Kim et al. (1996) allowed a job to start on the following (placement) machine before it is completed on the current (placement) machine, representing time lags appropriately because each job comprises a batch of identical CCs. They proposed several heuristics, including TSs and SAs, with the objective of minimizing mean tardiness. Schaller et al. (2000) modeled CC assembly as

$Fm \mid bs_{ijk}, prmu \mid C_{\max}$ with family setup times and required all CC types within each family to be assembled before starting the next family. They assumed that the families of different CC types were pre-determined and resorted to a heuristic. Logendran et al. (2003) modeled a CC assembly process as a two-machine flow shop and sequenced families of CC types using TS with the objective of minimizing mean flow time.

13.5. Combined lot sizing and SDS scheduling

Researchers have also investigated the problem of integrating lot-sizing and sequencing decisions in the flow shop. Sikora et al. (1996) considered a variation with limited intermediate buffer space and deadlines, and they studied the objectives of minimizing $C_{\max}$ and inventory holding costs. They integrated the Silver-Meal lot-sizing heuristic (Silver and Meal (1973)), which they modified to deal with lot splitting, with Palmer's flow shop heuristic (Palmer (1965)), which they augmented with an improvement procedure, and demonstrated the efficacy of their approach by scheduling an actual CC assembly line.

In another paper, Sikora (1996) presented a GA that used separate crossover and mutation operators for lot-sizing and sequencing decisions. He compared this GA (Sikora (1996)) with the integrated approach (Sikora et al. (1996)) and found that the GA that used a population size of 10 prescribed much better schedules with significantly less run time than the integrated approach. However, the performance of the GA was sensitive to the selection of parameter values and it was difficult to determine effective values.

CHAPTER XIV

THE JOB SHOP CONFIGURATION

The job shop scheduling problem – even without SDS – is NP-hard (Pinedo (2002)). Only a few papers have addressed job shop scheduling with SDS over the last decade. Kim and Bobrowski (1994) emphasized the impact of SDS on job shop scheduling performance.

This chapter reviews the limited research on job shop scheduling that involves SDS. Section 14.1 addresses optimizing and hybrid methods and Section 14.2 reviews heuristics. To our knowledge, over the last decade no research has been directed towards the combined lot-sizing and scheduling problem.

14.1. Optimizing and hybrid methods

Only one early paper investigated B&B in the job shop configuration; Gupta (1982) formulated a model for $Jm \mid recrc, s_{ijk} \mid \sum s_{ijk}$ but (as to be expected) B&B can solve only small instances because run time increases rapidly with problem size.

Luh et al. (1998) studied a facility that produced a variety of gas-insulated switch gears, each in small volume. Some stations required no setup; others required batch setups with longer setup time between dissimilar products than between similar products. They developed a MIP and a hybrid that used Lagrangian relaxation to relax machine capacity constraints, leading to several sub-problems for which they devised DP algorithms and heuristics with the goal of obtaining “near optimal” solutions.

14.2. Heuristics

GAs (Candido et al. (1998)), dispatching rules (Ovacik and Uzsoy (1994a)), and simulation (Kim and Bobrowski (1994)) models have been used to schedule job shops with SDS. Low (1995) related a heuristic and showed that it prescribed a more realistic solution than approaches that treat the problem as having either independent or no setups.

Zoghby et al. (2005) adapted the disjunctive graph model (Balas (1969)) to address reentry and introduced new features of this adapted disjunctive graph (Roy and Sussman (1964)) for $Jm \mid recrc, s_{ij} \mid \gamma$. They found that the traditional method (Balas (1969)) used to avoid infeasible solutions – reversing a disjunctive arc on the critical path – does not suffice, and presented an algorithm to remove infeasibilities. They further discussed potential applications of their results in meta-heuristics and decomposition methods (such as the shifting bottleneck method) to solve $Jm \mid recrc, s_{ij} \mid \gamma$. (The shifting-bottleneck procedure was originated by Adams et al. (1988) for $Jm \mid\mid C_{\max}$.)

To reschedule $Jm \mid r_j, s_{ij} \mid L_{\max}$, Artigues and Roubellat (2002) proposed a polynomial algorithm to insert a new operation into an existing schedule without changing previously scheduled operations. They suggested that their insertion algorithm could be used to improve the performance of some job-shop scheduling methods. For example, it could be used to generate a search neighborhood in a local search procedure simply by omitting a critical operation, using a simple heuristic to prescribe an initial schedule, and then reinserting the critical operation to construct a complete schedule.

To solve $Jm \mid s_{ijk} \mid C_{\max}$, Cheung and Zhou (2002) devised a heuristic, which combines a GA with a modified shortest-processing-time rule that accounts for SDS. Their computational results showed that this method can generate better solutions than the method of Choi and

Korkmaz (1997) but that it required much longer run times. Choi and Choi (2002) studied a variation of $Jm \mid s_{ijk} \mid C_{\max}$ that allowed alternative operations; they formulated a MIP with the objective of minimizing $C_{\max}$ and devised a local search procedure that substantially enhanced the performance of several greedy dispatching rules. Hertz and Widmer (1996) devised a TS for $Jm \mid s_{ijk} \mid C_{\max}$ and Sun et al. (2003) proposed a GA for the related $Jm \mid recrc, s_{ijk} \mid C_{\max}$.

With the goal of promoting on-time delivery in a semiconductor fabrication facility, Mason et al. (2002) investigated a specialization of $FJc \mid recrc, r_j, s_{ijk} \mid \sum w_j T_j$, which included a batching machine that processed several jobs simultaneously. They presented a disjunctive graph to model this complex job shop and proposed a modified shifting-bottleneck heuristic to deal with the added complexity of batching machines, parallel machines, SDS, and reentrant flow of jobs.

Kim and Bobrowski (1997) investigated the impact of uncertainty on sequencing decisions, considering stochastic processing, arrival, and SDS times while modeling SDS times as independent, normally distributed random variables. Their nine-machine job shop simulation tests showed that stochastic setup time had a negative impact on shop performance but did not diminish the advantages of sequencing rules (e.g., Wilbrecht and Prescott (1969) and Kim and Bobrowski (1994)) that deal explicitly with setup.

CHAPTER XV

PERSPECTIVES ON SDS SCHEDULING RESEARCH

This chapter provides perspectives on SDS scheduling research. We give two tables that provide broad perspectives. Table 14 gives a taxonomy of studies, categorized according to the four machine configurations and combined lot-sizing and scheduling. The three columns in Table 14 give the reference, designate the specific problem studied, and note the solution method, respectively. This taxonomy provides a useful structure by which studies on specific problem types can be related and a framework that shows relationships among studies.

Table 14. A taxonomy of SDS scheduling literature published over the last decade

Single machine	**Criterion (specializations)**	**Solution Method**
Monma and Potts (1989)	$1\|bs_{ij}\|C_{max}$, $1\|bs_{ij}\|L_{max}$, $1\|bs_{ij}\|\Sigma w_jC_j$, $1\|bs_{ij}\|\Sigma U_{j.}$	Complexity and DP formulation
Ghosh (1994)	$1\|bs_{ij}\|\Sigma(w_j)C_j$	Complexity and DP formulation
Chen (1997)	$1\|bs_{ij}\|\Sigma w_j'E_j+\Sigma w_j''T_j$ $1\|bs_{ij}\|\Sigma w_j'Ej+\Sigma w_j''T_j+\Sigma\beta_jd_j$ (common due date for each batch)	Complexity and DPs
Ozgur and Brown (1995)	$1\|bs_{ij}\|C_{max}$	Two-stage B&B/TSP heuristic
Spina et al. (2003)	$1\|s_{ij}\|C_{max}$	Hybrid: GA + Constraint Logic Programming
Rossetti and Stanford (2003)	$1\|bs_{ij}\|C_{max}$	Nearest-neighbor heuristic
Unal et al. (1997)	$1\|r_j,bs_{ij},\ \bar{d}_j\ \|C_{max}$ (rescheduling)	Polynomial algorithm (optimal)
	$1\|r_j,bs_{ij},\ \bar{d}_j\ \|\Sigma w_jC_j$ (rescheduling)	Proved strongly NP-hard; gave 2 heuristics
Rubin and Ragatz (1995)	$1\|s_{ij}\|\Sigma T_j$	GA
Tan and Narasimhan (1997a)	$1\|s_{ij}\|\Sigma T_j$	SA
Tan et al. (2000)	$1\|s_{ij}\|\Sigma T_j$	B&B, GA, SA and pairwise exchange
França et al. (2001)	$1\|s_{ij}\|\Sigma T_j$	Memetic (GA combined with local improvement)
Alidaee et al. (2001)	$1\|s_{ij}\|\Sigma T_j$	Generalized best-in greedy heuristic
Kim et al. (1995)	$1\|s_{ij}\|\Sigma w_jT_j$	Heuristic using ATCS rule and neural networks
Lee et al. (1997)	$1\|s_{ij}\|\Sigma w_jT_j$	Three-phase heuristic involving ATCS rule

Table 14. Continued

Single machine	**Criterion (specializations)**	**Solution Method**
Barnes and Vanston (1981)	$1\|s_{ij}\|\Sigma w_jC_j+\Sigma s_{ij}$	Hybrid B&B/DP
Laguna and Glover (1993)	$1\|s_{ij}\|\Sigma w_jC_j+\Sigma s_{ij}$	Integrated target analysis and TS
Feo et al. (1996)	$1\|s_{ij}\|\Sigma w_jC_j+\Sigma s_{ij}$	GRASP
Coleman (1992)	$1\|s_{ij}\|\Sigma w_j'E_j+\Sigma w_j''T_j$	MIP
Asano and Ohta (1996)	$1\|\ \overline{d}_j\ ,r_j,s_{ij}\|\Sigma E_j$	B&B incorporating dominance relation
Sun et al. (1999)	$1\|r_j,s_{ij}\|\Sigma w_jT_j^2$	Heuristic based on Lagrangian relaxation
Rabadi et al. (2003)	$1\|d,s_{ij}\|\Sigma E_j+\Sigma T_j$	B&B
Woodruff and Spearman (1992)	$1\|bs_{ij},\ \overline{d}_j\ \|\Pi'$	TS
Kolahan et al. (1995)	$1\|s_{ij}\|\mathrm{II}_1$	TS
Kolahan and Liang (1998)	$1\|s_{ij}\|\mathrm{II}_2$ with variable p_j	Adaptive TS
Laguna (1999)	$1\|s_{ij}\|\Pi'$	MIP and TS
Roslöf et al. (2002)	$1\|r_j,s_{ij}\|w'C_{max}+\Sigma w''T_j$	MIP solver combined with an iterative heuristic
Asano and Ohta (1999)	$1\|brkdwn,r_j,s_{ij}\|T_{max}$	B&B
Uzsoy et al. (1991)	$1\|prec,s_{ij}\|L_{max}$	B&B
Uzsoy et al. (1992)	$1\|prec,s_{ij}\|L_{max}$ $1\|prec,s_{ij}\|\Sigma U_j$	DP DP and heuristic
Ovacik and Uzsoy (1994b)	$1\|r_j,s_{ij}\|L_{max}$	Rolling horizon procedure
Tan and Narasimhan (1997b)	$1\|s_{ij}\|\mathrm{MO}(\Sigma w_jT_j\ \&\ \Sigma s_{ij})$	SA
Gupta and Sivakumar (2004)	$1\|s_{ij}\|\mathrm{MO}(\Sigma C_j/n,\ \Sigma T_j/n,\ \&\ C_{max})$	Simulation
Parallel Machines (PM)	**Criterion (specializations)**	**Solution Method**
Monma and Potts (1989)	$P2\|prmp,bs_{ij}\|C_{max}$, $P2\|prmp,bs_{ij}\|L_{max}$, $P2\|prmp,bs_{ij}\|\Sigma w_jC_j$, $P2\|prmp,bs_{ij}\|\Sigma U_j$.	Complexity
Cheng and Chen (1994)	$P2\|bs_{ij}\|\Sigma C_j$	Complexity
Ghosh (1994)	$Pm\|bs_{ij}\|\Sigma(w_j)C_j$	Complexity and DP formulation
Guinet (1993)	$Pm\|s_{ij}\|C_{max}$ $Pm\|s_{ij}\|\Sigma C_j$	MIP and a heuristic
França et al. (1996)	$Pm\|s_{ij}\|C_{max}$	TS-based, three-phase heuristic
Gendreau et al. (2001)	$Pm\|s_{ij}\|C_{max}$	Divide and merge heuristic
Mendes et al. (2002)	$Pm\|s_{ij}\|C_{max}$	TS-based and memetic heuristic
Kurz and Askin (2001)	$Pm\|r_j,s_{ij}\|C_{max}$	MIP and GA, multiple insertion (TSP), slicing (TSP) and multiple MULTI-FIT heuristics
Hurink and Knust (2001)	$Pm\|prec,s_{ij}\|C_{max}$	List scheduling algorithm
Yalaoui and Chu (2003)	$Pm\|s_{ij}\|C_{max}$ with job-splitting	B&B based heuristic
Schutten (1996)	$Pm\|r_j,s_{ij}\|\ \Re$	List scheduling algorithm
Weng et al. (2001)	$Rm\|s_{ij}\|\Sigma w_jC_j$	Proposed and tested seven heuristics
Lee and Pinedo (1997)	$Pm\|s_{ij}\|\Sigma w_jT_j$	3-phase ATCS heuristic to generate SA seed

Table 14. Continued

Parallel Machines (PM)	Criterion (specializations)	Solution Method
Park et al. (2000)	$Pm\|s_{ij}\|\Sigma w_jT_j$	Heuristic using ATCS rule and neural networks
Kim et al. (2003)	$Pm\|s_{ij}\|\Sigma w_jT_j$	TS-based Heuristic
Bilge et al. (2004)	$Qm\|r_j,s_{ijk}\|\Sigma T_j$	TS
Kim et al. (2002)	$Rm\|d,s_{ijk}\|\Sigma T_j$	SA
Heady and Zhu (1998)	$Pm\|s_{ij}\|\Sigma w_j'E_j+\Sigma w_j''T_j$	Greedy heuristic
Sivrikaya-Serifoglu and Ulusoy (1999)	$Qm\|r_j,s_{ijk}\|\Sigma w'E_j+\Sigma w''T_j$	Two GAs (one with crossover operator)
Balakrishnan et al. (1999)	$Qm\|r_j,s_{ijk}\|\Sigma w_j'E_j+\Sigma w_j''T_j$	MIP
Radhakrishnan and Ventura (2000)	$Pm\|s_{ij}\|\Sigma E_j+\Sigma T_j$	SA
Ovacik and Uzsoy (1993)	$Pm\|s_{ij}\|C_{max}$ $Pm\|s_{ij}\|L_{max}$	Worst-case error bound of list schedules
Ovacik and Uzsoy (1995)	$Pm\|r_j,s_{ij}\|L_{max}$	Rolling horizon procedure
Kim and Shin (2003)	$Rm\|r_j,s_{ij}\|L_{max}$	Restricted TS
Bitran and Gilbert (1990)	$Pm\|bs_{ij},prmp\|\Sigma bs_{ij}$	B&B/heuristic
Dhaenens-Flipo (2001)	$Qm\|\ \bar{d}\ , s_{ijk}\|$II, sum of setup and production-distribution costs	Heuristic
Hop and Nagarur (2004)	$Qm\|s_{ijk}\|$MO(workload balancing, board similarities, Σs_{ijk})	Composite GA
Arzi and Raviv (1998)	$Rm\|\ \tilde{r}_j\ ,\ \tilde{s}_{ijk}\ \|$throughput, Σs_{ijk} and WIP	Dispatching rules
Anglani et al. (2005)	$Pm\|\ \tilde{p}_j\ ,s_{ij}\|\Sigma s_{ij}$	Fuzzy mathematical programming + heuristic
Flow shop (FS)	**Criterion (specializations)**	**Solution Method**
Gupta (1986)	$Fm\|s_{ijk}\|\gamma$	Complexity
Gupta and Darrow (1986)	$F2\|s_{ijk}\|C_{max}$	Proved NP-complete and *prmu* not optimal
Srikar and Ghosh (1986)	$Fm\|prmu,s_{ijk}\|C_{max}$	MIP
Stafford and Tseng (1990)	$Fm\|prmu,s_{ijk}\|C_{max}$	Correction of Srikar and Ghosh's MIP
Simons (1992)	$Fm\|prmu,s_{ijk}\|C_{max}$	2 dispatching rules and 2 TSP-based heuristics
Das et al. (1995)	$Fm\|prmu,s_{ijk}\|C_{max}$	Saving index heuristic
Rios-Mercado and Bard (1998a)	$Fm\|prmu,s_{ijk}\|C_{max}$	B&C
Rios-Mercado and Bard (1998b)	$Fm\|prmu,s_{ijk}\|C_{max}$	GRASP
Rios-Mercado and Bard (1999a)	$Fm\|prmu,s_{ijk}\|C_{max}$	B&B heuristic with partial enumeration
Rios-Mercado and Bard (1999b)	$Fm\|prmu,s_{ijk}\|C_{max}$	Enhanced TSP-based heuristic
Schaller et al. (2000)	$Fm\|bs_{ijk},prmu\|C_{max}$	Heuristic
Stafford and Tseng (2002)	$Fm\|prmu,s_{ijk}\|C_{max}$ $Fm\|nwt,s_{ijk}\|C_{max}$	Two MIPs
Ruiz et al. (2005)	$Fm\|prmu,s_{ijk}\|C_{max}$	Two GAs
Kurz and Askin (2003)	$FFc\|s_{ijk}\|C_{max}$	Greedy & TSP-insertion heuristics, Johnson's rule
Kurz and Askin (2004)	$FFc\|s_{ijk}\|C_{max}$	Above + MIP and random keys GA

Table 14. Continued

Flow shop (FS)	Criterion (specializations)	Solution Method
Pugazhendhi et al. (2004)	$Fm\|s_{ijk}\|C_{max}$ $Fm\|s_{ijk}\|\Sigma w_jC_j$	Heuristic
Norman (1999)	$Fm\|block,prmu,s_{ijk}\|C_{max}$	TS
Bianco et al. (1999)	$Fm\|nwt,r_j,s_{ijk}\|C_{max}$	Greedy heuristics
Rajendran and Ziegler (1997)	$Fm\|prmu,s_{ijk}\|\Sigma w_jC_j$	Heuristic with an improvement scheme
Mendez et al. (2001)	$FFc\|M_j,r_j,s_{ijk}\|\Sigma E_j$ (batch plant)	Continuous-time MIP
Hui et al. (2000)	$FFc\|M_j,s_{ijk}\|\Sigma E_j+\Sigma T_j$ (batch plant)	Continuous-time MIP
Parthasarathy and Rajendran (1997a)	$Fm\|prmu,s_{ijk}\|\max w_jT_j$ $Fm\|prmu,s_{ijk}\|\Sigma w_jT_j$	SA
Parthasarathy and Rajendran (1997b)	$Fm\|prmu,s_{ijk}\|(\Sigma w_jT_j)/n$	SA
Rajendran and Ziegler (2003)	$Fm\|prmu,s_{ijk}\|\Sigma w_jC_j+\Sigma w'_jT_j$	Heuristic with an improvement scheme
Pearn et al. (2004)	$FFc\|bs_{ijk}, \overline{d}_j, recrc\|$max total machine workload	Network algorithms
Ovacik and Uzsoy (1994a)	$FFc\|recrc,s_{ijk}\|L_{max}$	Dispatching rule based heuristic
Demirkol and Uzsoy (2000)	$FFc\|recrc,s_{ijk}\|L_{max}$	Decomposition method
Hwang and Sun (1997)	$F2\|recrc,s_{ij1}\|C_{max}$	DP
Hwang and Sun (1998)	$F2\|recrc,s_{ij1}\|C_{max}$	DP and GA
Sun and Hwang (2001)	$F2\|s_{ij2}\|C_{max}$	DP and GA
Maimon et al. (1993)	$F2\|s_{ijk}\|$throughput and WIP	Two scheduling methods
Logendran et al. (2003)	$F2\|bs_{ijk}\|(\Sigma w_jT_j)/n$	Lower bound and TS
Kim et al. (1996)	$Fm\|s_{ijk}\|(\Sigma T_j)/n$ (time-lag)	Heuristics (e.g., TS, SA)
Job shop	**Criterion (specializations)**	**Solution Method**
Candido et al. (1998)	$Jm\|s_{ij}\|\ \Re$	GA-based heuristic
Hertz and Widmer (1996)	$Jm\|s_{ijk}\|C_{max}$	TS
Choi and Korkmaz (1997)	$Jm\|s_{ijk}\|C_{max}$	MIP and a heuristic
Cheung and Zhou (2002)	$Jm\|s_{ijk}\|C_{max}$	Hybrid GA heuristic with SPTS rule
Choi and Choi (2002)	$Jm\|s_{ijk}\|C_{max}$	MIP and a local search scheme
Sun et al. (2003)	$Jm\|recrc,s_{ijk}\|C_{max}$	Hybrid GA and heuristics
Ovacik and Uzsoy (1994a)	$Jm\|s_{ijk}\|L_{max}$	Dispatching rule based heuristic
Artigues and Roubellat (2002)	$Jm\|r_j,s_{ij}\|L_{max}$ (insertion problem)	Insertion algorithm
Gupta (1982)	$Jm\|recrc,s_{ijk}\|\Sigma s_{ijk}$	B&B
Low (1995)	$Jm\|r_j,s_{ijk}\|$mean flow time, mean tardiness, mean machine idle	Heuristic
Kim and Bobrowski (1994)	$Jm\|s_{ijk}\|\gamma$	Simulation
Zoghby et al. (2005)	$Jm\|recrc,s_{ij}\|\gamma$	Meta-heuristics and disjunctive graph
Kim and Bobrowski (1997)	$Jm\|\widetilde{s}_{ijk}\|\gamma$ (stochastic $\widetilde{r}_j$, $\widetilde{p}_j$, $\widetilde{s}_{ijk}$)	Dispatching rules
Luh et al. (1998)	$FJc\|block,bs_{ijk},r_j\|\Sigma w_j'E_j^2+\Sigma w_j''T_j^2$	MIP and hybrid: Lagrangian relax, DP + heuristic
Mason et al. (2002)	$FJc\|recrc,r_j,s_{ij}\|\Sigma w_jT_j$ (batch plant)	Modified shifting bottleneck heuristic

Table 14. Continued

Combined lot-sizing	Criterion (specializations)	Solution Method
Dobson (1992)	Π′ (ELSP: single machine)	Lagarangian relaxation based heuristic
Wagner and Davis (2002)	Π′ (ELSP: single machine)	Heuristic
Miller et al. (1999)	min sum of setup, inventory and backlog cost (CLSP: single machine)	MIP and GA with hill-climbing local search
Haase (1996)	Π′ (CLSP: single machine)	Heuristic priority rule
Haase and Kimms (2000)	Π′ (CLSP: single machine)	MIP and B&B with a tailor-made enumeration
Gupta and Magnusson (2005)	Π′ (CLSP: single machine)	MIP and heuristic
Fleischmann (1994)	Π′ (DLSP: single machine)	Formulation (TSP with time windows) + heuristic
Salomon et al. (1997)	Π′ (DLSP: single machine)	DP
Jordan and Drexl (1998)	Π′ (DLSP: single machine)	B&B
Meyr (2000)	Π′ (GLSP: single machine)	MIP & dual re-optimization + heuristic
Meyr (2002)	min sum of setup, holding and production costs (non-identical parallel machines)	Dual re-optimization + heuristic (SA or threshold accepting)
Kang et al. (1999)	min sum of setup, holding and production costs minus sales revenue (parallel machines)	Hybrid column generation\B&B + heuristic
Sikora et al. (1996)	$Fm\|block, \overline{d}_j, s_{ijk}\|$bi-objective ($C_{max}$ & min holding cost)	Integrated heuristic approach
Sikora (1996)	Same as above	GA with crossover and mutation operators

Π_1 – sum of setup, defective part, tool and machining costs
Π_2 - sum of earliness and tardiness penalties and compression and extension costs
MO – multi-objective

Table 15 lends further perspective by tallying the number of studies cited that deal with each combination of methodology (Column 1) and machine configuration (Columns 2-5). The first methodology involves formulation of SDS problems. Most formulations are MIP models; few papers report computational experience because run times tend to be excessive for these NP-hard problems, even for instances of modest size. The second methodology, which involves analyzing the complexity of different types of problems, is typically based on a DP formulation (e.g., Monma and Potts (1989), Ghosh (1994), and Chen (1997)).

Table 15. Summary of research methodologies for SDS scheduling

Methodologies		Single machine	Parallel machine	Flow shop	Job shop
Formulation		2 (4)	4	6	3
Complexity analysis		3	3	1	
Optimizing methods					
B&B		6 (2)			1
B&C				1	
DP		3 (1)		3	
MIP solver		1	1	5	
Hybrid optimizing-and-heuristic methods					
Based on B&B, DP, or MIP solver		2	2 (1)	1	1
Heuristics					
Meta-heuristic	GA (hybrid GA)	4 (1)	4	4 (1)	3
	SA	3 (1)	4 (1)	3	
	TS	5	5	3	1
	GRASP	1		1	
Methods based on TSP heuristics		1	1	4	
Greedy algorithm		1	1	3	
Decomposition		1	1	1	1
Dispatching rules		2	3	2	3
Simulation		1			1
List scheduling			3		

Note: The number in each cell gives the number of studies that applied the methodology to the machine configuration excluding combined lot-sizing, which is tallied as the number in parentheses.

The third methodology, optimizing methods, includes B&B, B&C, DP, and MIP solvers. Most papers have focused on the single machine configuration; few have studied the parallel machine and job shop configurations. B&B has been used extensively to optimize the single machine configuration; only one early study (Gupta (1986)) applied B&B to the job shop configuration and none have applied B&B to the parallel machine and flow shop configurations. Only one paper, Rios-Mercado and Bard (1998a), has applied B&C to SDS; it addressed a flow shop problem, $Fm \mid prmu, s_{ijk} \mid C_{\max}$. DP has been applied rather extensively to single machine and flow shop configurations but not to parallel machine and job shop configurations. MIP

solvers (e.g., LINDO (Coleman (1992), Balakrishnan et al. (1999), and Stafford and Tseng (2002)) and CPLEX (Mendez et al. (2001))) have been used primarily in application to the flow shop configuration but they can only solve problems of small-to-modest size.

The fourth methodology, hybrid methods, combines an optimizing method (e.g., B&B, DP, or MIP solvers) and a heuristic, which either prescribes a solution found early in the search process or resolves some portion of the problem. For example, Ozgur and Brown (1995) and Bitran and Gilbert (1990) sequenced jobs in each family using heuristics and sequenced families using B&B for $1|bs_{ij}|C_{\max}$ and $Pm|bs_{ij}, prmp|\sum s_{ij}$, respectively; Roslöf et al. (2002) augmented a MIP solver with an iterative heuristic for $1|r_j, s_{ij}|w'C_{\max}+\sum w''T_j$; and Rios-Mercado and Bard (1999a) used B&B with partial enumeration to approximate $Fm|prmu, s_{ijk}|C_{\max}$. Actually, the rolling horizon method in Ovacik and Uzsoy (1994b) belongs to this class; it uses B&B to determine the next job to process and applies it in a heuristic fashion, solving the dynamic scheduling problem by applying B&B at each time a scheduling decision must be made.

The fifth methodology, heuristics, is rather expansive because heuristics are motivated by the inherent difficulty of SDS scheduling problems. Although heuristics may provide good approximate solutions in reasonable run time, few give performance guarantees for problems involving SDS.

Table 15 shows that meta-heuristics have been widely used for all machine configurations. In particular, GA and TS have been the most favored approaches for a variety of objective functions, including makespan ($C_{\max}$) as well as those related to due dates (e.g., $L_{\max}$, $\sum T_j$, $\sum w_j T_j$). Interestingly, França et al. (1996) and Kim et al. (2003) reported that the performance of TS improved with the number of machines in the parallel machine configuration.

SA places third, close behind GA and TS. In contrast, researchers have applied GRASP in only two recent papers.

Since SDS is intimately related to TSP - for example, $1|s_{ij}|C_{\max}$ is, in fact, TSP - heuristics based on TSP algorithms have been evolved for the single machine (e.g., Ozgur and Brown (1995) for $1|bs_{ij}|C_{\max}$), parallel machine (e.g., Kurz and Askin (2001) for $Pm|r_j,s_{ij}|C_{\max}$), and flow shop (e.g., Rios-Mercado and Bard (1999b) for $Fm|s_{ijk},prmu|C_{\max}$) configurations. A feature that is common to these papers is that they have all addressed the objective of minimizing $C_{\max}$.

Other heuristics – greedy algorithm; decomposition; dispatching rules; simulation; list scheduling; and heuristics based on TSP algorithms – have all been used, but by fewer researchers. Decomposition divides an intractable problem into smaller, less challenging sub-problems, develops solutions for the sub-problems, and assembles them into a schedule for the original problem. An effective implementation requires fast procedures to obtain high-quality solutions to the sub-problems. Recent applications of decomposition include rolling horizon heuristics for $1|s_{ij}|L_{\max}$ (Ovacik and Uzsoy (1994b)) and $Pm|s_{ij}|L_{\max}$ (Ovacik and Uzsoy (1995)) and a modified shifting bottleneck procedure for $FJc|recrc,r_j,s_{ij}|\sum w_jT_j$ (Mason et al. (2000)). Dispatching rules, which are typically tested using a simulation model, are popular because they reflect how decisions are made in many practical cases. Neural networks have been used to determine the parameters required by ATCS rules (Kim et al. (1995) for $1|s_{ij}|\sum w_jT_j$ and Park et al. (2000) for $Pm|s_{ij}|\sum w_jT_j$). List scheduling algorithms offer an advantage in that they have been shown to provide worst-case error bounds for $Pm|s_{ij}|C_{\max}$ and $Pm|s_{ij}|L_{\max}$ (Ovacik and Uzsoy (1993)) and dominant schedules for $Pm|r_j,s_{ij}|\Re$ (Schutten

(1996)) but it is known that they cannot generate a set of dominant schedules for $Pm \mid prec, s_{ij} \mid C_{\max}$ (Hurink and Knust (2001)). We present our conclusions in the next chapter.

CHAPTER XVI

CONCLUSIONS AND FUTURE RESEARCH

16.1. Conclusions on the RCSP research

To solve RCSP, this dissertation proposes TSA, which comprises two phases: a one-time preliminary phase (stages 1 and 2) and an iterative solution phase (stage 3), as shown in Figure 1. TSA requires less computational effort for each subsequent solution than the first-time solution and is especially suitable for solving RCSP repeatedly, for example, as a subproblem in CG and CG/B&B. Computational results demonstrate the effectiveness of TSA and show that it outperforms a method we devised for benchmarking purpose (i.e., LSA) in the context of CG. TSA incorporates several specialized algorithms for dealing with particular issues (e.g., preprocessing, reoptimizing and fixed arcs); computational results demonstrate the effectiveness of these algorithms.

TSA is a unified approach comprising three stages. By applying adaptations of stage 1, it can solve RCSP, SPPRW and SPRCRW. Stages 2 and 3 are applicable to each of these problems. By incorporating a classical unconstrained *k*-SPP algorithm in stage 3, TSA can solve RC*k*SP on an acyclic graph. Further, TSA can be applied to MMCKP, recasting it as RCSP on the MMCKP-graph. Since TSA is suitable for solving RCSP repeatedly, each of these extensions is suitable for solving the corresponding problem repeatedly.

This dissertation research shows that algorithms for solving subproblems in CG (e.g., RCSP) should explicitly consider issues related to repeated solution. If possible, a preliminary phase can be used to reduce the computational burden incurred by repeated solutions. Moreover, if the algorithms are used to solve a subproblem in CG/B&B, then effective methods for dealing with fixed variables should also be considered when designing the algorithms. For CG/B&B, the

branching strategy of B&B should be considered together with algorithms for solving subproblem(s) to obtain good overall performance. For example, if the subproblem is RCSP and we use TSA to solve it, it is appropriate to choose a branching variable whose corresponding arc has a large span (see Remark 4) and it is not good to choose the one that has an associated arc that can be relegated to set $\hat{F}_0$ (see Remark 5).

16.2. Conclusions on the SDS scheduling literature review

This dissertation contributes by achieving its purpose of reviewing SDS scheduling research as a guide for future research. It also achieves its objectives, providing (i) an overview with emphasis on recent results, (ii) an integrated view of lot-sizing and SDS scheduling, (iii) a perspective of this line of research, and (iv) fertile opportunities for future research (Section 16.3). The perspective is enhanced by a taxonomy that classifies research according to machine configuration with an emphasis on problem type studied and by tallying of studies according to the methodology applied to the machine configuration. Overall, the perspective emphasizes that SDS is relevant to virtually all machine configurations. We address the fourth objective of this part of dissertation research in the next section, discussing fertile research opportunities.

16.3. Future research on SDS scheduling problems

The taxonomy (Table 14) shows that SDS is attracting an increasing amount of interest and Table 15 shows that a variety of methods have been considered for each machine configuration. Despite this amount of attention, no solution approach is widely recognized as providing superior capability to resolve problems in this class. We suggest that research would be further stimulated by establishing a set of test instances that would allow rigorous comparison of solution methods. This section discusses research opportunities relative to optimizing methods,

heuristics, objectives related to due dates, bi- or multi-objectives, machine configurations, combined lot sizing and scheduling, rescheduling, stochastic scheduling, and CC assembly.

16.3.1. Optimizing methods

This review shows clearly that optimizing methods for SDS scheduling have not enjoyed the steady - and sometimes dramatic – progression of improvements that have enhanced integer-programming methodologies over the last decade. Future research could bridge this gap, more fully exploring opportunities to adapt methods like B&P and B&C to SDS scheduling.

16.3.2. Heuristics

Most existing methods are heuristics. Although they may provide good approximate solutions in reasonable run time, few give performance guarantees. Researchers have proposed numerous heuristics with little theoretical underpinnings. A significant challenge for future research is, then, to analyze worst-case error bounds for heuristics. Several studies have proposed approaches to determine parameter values upon which heuristics depend (e.g., TS and SA (Kim et al. (1996)), GA (Sikora (1996)), ATCS rule (Lee et al. (1992), Kim et al. (1995), Park et al. (2000)), and priority rule (Haase (1996))) but future research is needed to establish an integrated knowledge base for specifying effective parameter values. In addition, future research should be directed toward devising formal structures that incorporate local search methods to exploit the diversification of GAs in combination with intensification of local searches (e.g., Miller et al. (1999), Mendes et al. (2002), França et al. (2001), Moscato (1989, 1999), Spina et al. (2003), and Ruiz et al. (2005)).

16.3.3. Objectives related to due dates

Most papers address the objective of minimizing $C_{\max}$, especially for flow shops and job shops: this review cites 22 papers that deal with $C_{\max}$ and only 7 that address due-date objectives for

the flow shop and 7 papers that deal with $C_{\max}$ compared with only 3 papers that address due-date objectives for the job shop. Motivated by the need for just-in-time production, a few researchers have investigated the total (weighted) earliness and tardiness objective for the single machine (Coleman (1992), Rabadi et al. (2004)), parallel machine (Heady and Zhu (1998), Radhakrishnan and Ventura (2000), Sivrikaya-Serifoglu and Ulusoy (1999), Balakrishnan et al. (1999)), flow shop (Hui et al. (2000), Rajendran and Ziegler (2003)), and job shop (Luh et al. (1998)) configurations. Fertile opportunities are thus available for future research to address due-date related objectives.

16.3.4. Bi- or multi-objectives

In many real-world applications, it may be necessary to consider several objectives simultaneously and a schedule that is acceptable relative to one criterion may be unacceptable relative to another. Relatively few papers have dealt with bi- or multi-objectives (e.g., single machine (Tan and Narasimhan (1997b), Gupta and Sivakumar (2004)), parallel machines (Hop and Nagarur (2004)), and flow shop (Sikora (1996), Sikora et al. (1996)), leaving further opportunities for the future.

16.3.5. Machine configurations

The single machine configuration has received the bulk of attention due to its relative simplicity. A modest amount of research has addressed the parallel machine and flow shop configurations. Most flow shop research has dealt with permutation schedules (this review cites only one paper (Pugazhendhi et al. (2004)) that dealt with non-permutation schedules) but they are not necessarily optimal for the flow shop with SDS (except for the no-wait flow shop (Gupta (1986))); this issue poses research challenges. Relatively few studies have investigated the job

shop configuration and, to our knowledge, none has studied its combined lot-sizing and scheduling problem with SDS.

16.3.6. Combined lot-sizing and scheduling

Most research related to combined lot-sizing and scheduling problems with SDS has focused on the single machine configuration. Research on other configurations is sparse. Only two papers (Kang et al. (1999), Meyr (2002)) studied the parallel machine configuration and two others (Sikora (1996), Sikora et al. (1996)) studied the flow shop configuration; to our knowledge, none has considered the job shop configuration. Most research on combined lot-sizing and scheduling has not considered backordering; one exception relates to the single machine configuration (Miller et al. (1999)).

16.3.7. Rescheduling

As part of a current trend, two papers have dealt with rescheduling problems in dynamic environments (e.g., Artigues and Roubellat (2002) for $Jm \mid r_j, s_{ijk} \mid L_{\max}$ and Unal et al. (1997) for $1 \mid \overline{d}_j, r_j, s_{ij}, \mid \sum w_j C_j$ or $C_{\max}$). Rescheduling is an important issue for all machine configurations and objectives since it is important to be able to respond to unforeseen events.

16.3.8. Stochastic scheduling

Few researchers have considered stochastic scheduling problems with SDS. Three papers have dealt with uncertain processing, setup and/or arrival times: two (Arzi and Raviv (1998), Anglani et al. (2005)) dealt with the parallel machine configuration; and one (Kim and Bobrowski (1997)), with the job shop configuration. Thus, stochastic scheduling with SDS is another fertile opportunity for future research.

16.3.9. CC Assembly

This review discusses four papers that model CC assembly as a flow shop; one, as a set of parallel machines; and one, as a single machine. In all cases, SDS times are substantial, emphasizing the need for effective solution methods. Optimizing methods have not been investigated extensively for CC assembly, even though the CC industry is in need of effective scheduling methods. In addition, little progress has been made on stochastic models for scheduling CC assembly.

REFERENCES

Adams, J., Balas, E. and Zawack, D. (1988) The shifting bottleneck procedure for job shop scheduling. *Management Science*, **34**, 391-401.

Aldowaisan, T. and Allahverdi, A. (2003) New heuristics for no-wait flow Shops to minimize makespan. *Computers & Operations Research*, **30**, 1219-1231.

Alidaee, B., Kochenberger, G.A. and Amini, M.M. (2001) Greedy solutions of selection and ordering problems. *European Journal of Operational Research*, **134**, 203-215.

Allahverdi, A., Gupta, J.N.D. and Aldowaisan, T. (1999) A review of scheduling research involving setup considerations. *Omega*, **27**(2), 219-239.

Aneja, Y.P., Aggarwal, V. and Nair, K.P.K. (1983) Shortest chain subject to side constraints. *Networks*, **13**, 295-302.

Anglani, A., Grieco, A., Guerriero, E. and Musmanno, R. (2005) Robust scheduling of parallel machines with sequence-dependent set-up costs. *European Journal of Operational Research*, **161**, 704-720.

Artigues, C. and Roubellat, O. (2002) An efficient algorithm for operation insertion in a multi-resource job-shop schedule with sequence-dependent setup times. *Production Planning & Control*, **13**(2), 175-186.

Arzi, Y. and Raviv, D. (1998) Dispatching in a workstation belonging to a re-entrant production line under sequence-dependent set-up times. *Production Planning & Control*, **9**(7), 690-699.

Asano, M., and Ohta, H. (1996) Single machine scheduling using dominance relation to minimize earliness subject to ready and due times. *International Journal of Production Economics*, **44**(1-2), 35-43.

Asano, M. and Ohta, H. (1999) Scheduling with shutdowns and sequence dependent setup times. *International Journal of Production Research*, **37**(7), 1661-1676.

Avella, P., Boccia, M. and Sforza, A. (2004) Resource constrained shortest path problems in path planning for fleet management. *Journal of Mathematical Modelling and Algorithms*, **3**, 1-17.

Aytug, H., Lawley, M.A., McKay, K., Mohan, S. and Uzsoy, R. (2005) Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, **161**, 86-110.

Babonneau, F., Merle, O.du and Vial, J.-P. (2004) Solving large scale linear multicommodity flow problems with an active set strategy and Proximal-ACCPM. *Working Paper*.

Balakrishnan, N. (1993) Simple heuristics for the vehicle routing problem with soft time windows. *Journal of Operational Research Society*, **44**, 279-287.

Balakrishnan, N., Kanet, J.J. and Sridharan, S.V. (1999) Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers & Operations Research*, **26**(2), 127-141.

Balas, E. (1969) Machine sequencing via disjunctive graphs: an implicit enumeration approach. *Operations Research*, **17**, 941-957.

Bard, J.F. and Miller, J.L. (1989) Probabilistic shortest path problems with budgetary constraints. *Computers & Operations Research*, **16**(2), 145-159.

Barnes, J.W. and Vanston, L.K. (1981) Scheduling jobs with linear delay penalties and sequence dependent setup costs. *Operations Research*, **29**, 146-154.

Barrett, C., Jacob R. and Marathe M. (2000) Formal-Language-constrained path problems. *SIAM Journal on Computing*, **30**(3), 809-837.

Bazaraa, M.S., Jarvis, J.J. and Sherali, H.D. (1990) *Linear Programming and Network Flows*, John Wiley & Sons, New York, 320-350.

Beasley, J.E. and Christofides, N. (1989) An algorithm for the resource constrained shortest path problem. *Networks*, **19**, 379-394.

Bellman R.E. (1958) On a routing problem. *Quart. Appl. Math*, **16**, 87-90.

Berman, O., Einav, D. and Handler, G. (1990) The constrained bottleneck problem in network. *Operations Research*, **38**(1), 178-181.

Bianco, L., Dell'Olmo, P. and Giordani, S. (1999), Flowshop no-wait scheduling with sequence-dependent setup times and release dates. *INFOR*, **37**(1), 3-19.

Bilge, U., Klrac, F., Kurtulan, M. and Pekgü, P. (2004) A tabu search algorithm for parallel machine total tardiness problem. *Computers & Operations Research*, **31**(3), 397-414.

Bitran, G.R. and Gilbert S.M. (1990) Sequencing production on parallel machines with two magnitudes of sequence-dependent setup cost. *Journal of Manufacturing Operations Management*, **3**, 24-52.

Blocher, J.D., Chand, S. and Sengupta, K. (1999) The changeover scheduling problem with time and cost considerations: analytical results and a forward algorithm. *Operations Research*, **47**(4), 559-569.

Bodin, L.D., Golden, B.L., Assad, A.A. and Ball, M.O. (1982) Routing and scheduling of vehicles and crews: the state of the art. *Computers & Operations Research*, **10**, 63-211.

Bramel, J. and Simchilevi, D. (1996) Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows. *Operations Research*, **44**, 501-509.

Buriol, L.S., Resende, M.G.C. and Thorup, M. (2003) Speeding up dynamic shortest path algorithms. *AT&T labs Research Technical Report*, TD-5RJ8B, Florham Park, NJ.

Candido, M.A.B., Khator, S.K. and Barcia, R.M. (1998) A genetic algorithm based procedure for more realistic job shop scheduling problems. *International Journal of Production Research*, **36**(12), 3437-3457.

Cattrysse, D., Salomon, M., Kuik, R. and van Wassenhove L.N. (1993) A dual ascent and column generation heuristic for the discrete lotsizing and scheduling problem with setup times. *Management Science*, **39**(4), 477-486.

Chen, G.-H. and Hung, Y.-G. (1994) Algorithms for the constrained quickest path problem and the enumeration of quickest paths. *Computers & Operations Research*, **21**(2), 113-118.

Chen, Y.L. and Tang, K. (1997) Shortest paths in time-schedule networks. *International Journal of Operations and Quantitative Management*, **3**, 157-173.

Chen, Y.L. and Yang, H.H. (2000) Shortest paths in traffic-light networks. *Transportation Research Part B*, **34**, 241-253.

Chen, Y.L. and Yang, H.H. (2003) Minimization of travel time and weighted number of stops in a traffic-light network. *European Journal of Operational Research*, **144**, 565-580.

Chen, Z.-L. (1997) Scheduling with batch setup times and earliness-tardiness penalties. *European Journal of Operational Research*, **96**, 518-537.

Cheng, T.C.E. and Chen, Z.-L. (1994) Parallel machine scheduling with batch setup times. *Operations Research*, **42**(6), 1171-1174.

Cheng, T.C.E., Gupta, J.N.D. and Wang, G. (2000) A review of flowshop scheduling with setup times. *Production and Operations Management*, **9**(3), 262-282.

Cheung, W. and Zhou H. (2002) Using generic algorithms and heuristics for job shop scheduling with sequence-dependent setup times. *Annals of Operations Research*, **107**, 65-81.

Choi, I. and Choi, D. (2002) A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups. *Computers & Industrial Engineering*, **42**(1), 43-58.

Choi, I.C. and Korkmaz O. (1997) Job shop scheduling with sequence-dependent setups. *Annals of Operations Research*, **70**, 155-170.

Choi, I., Kim, S. and Kim, H. (2003) A genetic algorithm with a mixed region search for the asymmetric traveling salesman problem. *Computers & Operations Research*, **30**(5), 773-786.

Christofides, N., Mingozzia, A. and Toth, P. (1981) Exact algorithms for the vehicle routing problem, based on spanning trees and shortest path relaxations. *Mathematical Programming*, **20**, 255-282.

Coffman, Jr., E.G., Garey, M.R. and Johnson, D.S., (1978) An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, **7**, 1-17.

Coleman, B.J. (1992) A simple model for optimizing the single machine early/tardy problem with sequence-dependent setups. *Production and Operations Management*, **1**(2), 225-228.

Dantzig, G.B. and Wolfe, P. (1960) Decomposition principle for linear programs. *Operations Research*, **8**, 101-111.

Das, S.R., Gupta, J.N.D. and Khumawala, B.M. (1995) A saving index heuristic algorithm for flowshop scheduling with sequence-dependent setup times, *Journal of the Operational Research Society*, **46**, 1365-1373.

Daul, G. and Gouveia, L. (2004) On the directed hop-constrained shortest path problem. *Operations Research Letters*, **32**, 15-32.

Delporte, C.M. and Thomas, L.J. (1977) Lot sizing and sequencing for *N* products on one facility. *Management Science*, **23**(10), 1070-1079.

Demetrescu, C. (2001) *Fully Dynamic Algorithm for Path Problems on Directed Graphs*. PhD thesis, Department of Computer and Systems Science, University of Rome "La Sapienza".

Demirkol, E. and Uzsoy, R. (2000) Decomposition methods for reentrant flow shops with sequence-dependent setup times. *Journal of Scheduling*, **3**(3), 155-177.

Deo, N. and Pang, C. (1984) Shortest path algorithms: taxonomy and annotation. *Networks*, **14**, 275-323.

Desaulniers, G., Desrosiers, J., Dumas, Y., Marc, S., Rioux, B., Solomon, M.M. and Soumis, F. (1997) Crew pairing at Air France, *European Journal of Operational Research*, **97**, 245-259.

Desaulniers, G., Desrosiers, J. and Solomon, M.M. (2002) Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. in *Essays and Surveys in Metaheuristics*, Ribeiro, C.C. and Hansen, P. (eds), Kluwer, Norwell, MA, 309-324.

Desrochers, M. (1988) An algorithm for the shortest path problem with resource constraints. *Technical Report* G-88-27, GEARD, Ecole des H.E.C., Montreal, Canada.

Desrochers, M., Desrosiers, J. and Solomon, M. (1992) A new optimization algorithm for the vehicle routing problem with time windows, *Operations Research*, **40**(2), 342-354.

Desrochers, M. and Soumis, F. (1988a) A reoptimization algorithm for the shortest path problem with time windows. *European Journal of Operational Research*, **35**, 242-254.

Desrochers, M. and Soumis, F. (1988b) A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR*, **26**, 193-214.

Desrochers, M. and Soumis, F. (1989) A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, **23**, 1-13.

Desrosiers, J., Dumas, Y. and Soumis, F. (1988) The multiple vehicle dial-a-ride problem. in *Computer-Aided Transit Scheduling*, Daduda, J.R. and Wren, A. (eds), Lecture Notes in Economics and Mathematical System 308, Springer, Berlin, 15-27.

Desrosiers, J., Dumas, Y., Solomon, M.M. and Soumis, F. (1993) *Time Constrained Routing and Scheduling*, North-Holland, Amsterdam, The Netherlands.

Desrosiers, J., Soumis, F. and Desrochers, M. (1984) Routing with time windows by column generation. *Networks*, **14**, 545-565.

Dhaenens-Flipo, C. (2001) A bicriterion approach to deal with a constrained single-objective problem. *International Journal of Production Economics*, **74**, 93-101.

Dijkstra, E.W. (1959) A note on two problems in connection with graphs. *Numerische Mathematik*, **1**, 269-271.

Dobson, G. (1992) The cyclic lot scheduling problem with sequence-dependent setups. *Operations Research*, **40**(4), 736-749.

Dobson, G., Karmarkar, U.S. and Rummel, J.L. (1987) Batching to minimize flow times on one machine. *Management Science*, **33**, 784-799.

Dolgui, A., Guschinsky, N. and Levin, G. (2004) A special case of transfer lines balancing by graph approach. *European Journal of Operational Research*, In press.

Drexl, A. and Kimms, A. (1997) Lot sizing and scheduling – survey and extensions. *European Journal of Operational Research*, **99**, 221-235.

Dror, M. (1994) Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, **42**(5), 977-978.

Dudzinski, K. and Walukiewicz, S. (1987) Exact methods for knapsack problem and its generalizations. *European Journal of Operational Research*, **28**, 3-21.

Dumas, Y., Desrosiers, J. and Soumis, F. (1991) The pickup and delivery problem with time windows. *European Journal of Operational Research*, **54**, 7-22.

Dumitrescu, I. and Boland, N. (2001) Algorithm for the weight constrained shortest path problem. *ITOR*, **8**, 15-30.

Dumitrescu, I. and Boland, N. (2003) Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, **42**(3), 135-153.

Elimam, A.A. and Kohler, D. (1997) Two engineering applications of a constrained shortest-path model. *European Journal of Operational Research*, **103**, 426-438.

Eppstein, D. (1998) Finding the *k* shortest paths. *SIAM Journal on Computing*, **28**(2), 652-673.

Feillet, D., Dejax, P., Gendreau, M. and Gueguen, C. (2004) An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. *Networks*, **44**(3), 216-229.

Feo, T.A., Bard, J.F. and Holland, S.D. (1995) Facility-wide planning and scheduling of printed wiring board assembly. *Operations Research*, **43**(2), 219-230.

Feo, T.A., Sarathy, K. and McGahan, J. (1996) A grasp for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers & Operations Research*, **23**(9), 881-895.

Fisher, M.L. (1981) The Lagrangian relaxation method for solving integer programming problems. *Management Science*, **27**, 1-18.

Fleischmann, B. (1990) The discrete lot-sizing and scheduling problem. *European Journal of Operations Research*, **44**, 337-348.

Fleischmann, B. (1994) The discrete lot-sizing and scheduling problem with sequence-dependent setup costs. *European Journal of Operational Research*, **75**(2), 395-404.

Fleischmann, B. and Meyr, H. (1997) The general lotsizing and scheduling problem. *OR Spektrum*, **19**(1), 11-21.

Ford, L., Jr. (1956) *Network Flow Theory*. The Rand Corporation, Santa Monica, CA, P-923.

Ford, L.R. and Fullkerson, D.R. (1958) A suggested computation for maximal multi-commodity network flows. *Management Science*, **5**, 97-101.

Fox, B.L. (1978) Data structure and computer science techniques in operations research. *Operations Research*, **26**, 686-717.

França, P.M., Gendreau, M., Laporte, G. and Muller, F.M. (1996) A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, **43**(2-3), 79-89.

França, P.M., Mendes, A. and Moscato, P. (2001) A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research*, **132**, 224-242.

Fujishige, S. (1981) A note on the problem of updating shortest paths. *Networks*, **11**, 317-319.

Gallo, G. (1980) Reoptimization procedures on shortest path problems. *Riv. Mat. Sci. Econom. Social*, **3**, 3-13.

Gallo, G. and Pallottino, S. (1986) Shortest path methods: A unifying approach. *Mathematical Programming Study*, **26**, 38-64.

Gamache, M., Grimard, R. and Cohen, P. (2005) A shortest-path algorithm for solving the fleet management problem in underground mines. *European Journal of Operational Research*, **166**, 497-506.

Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

Gendreau, M., Laporte, G. and Guimarces, E.M. (2001) A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, **133**(1), 183-189.

Ghosh, J.B. (1994) Batch scheduling to minimize total completion time. *Operations Research Letter*, **16**, 271-275.

Gilmore, P.C. and Gomory, R.E. (1961) A linear programming approach to the cutting-stock problem. *Operations Research*, **9**, 849-859.

Glover, F. (1989) Tabu search—Part I. *ORSA Journal on Computing*, **1**, 190-206.

Glover, F. (1990) Tabu search—Part II. *ORSA Journal on Computing*, **2**, 40-42.

Glover, F., Klingman D.D. and Phillips, N.V. (1985a) A new polynomials bounded shortest path algorithm. *Operations Research*, **33**(1), 65-73.

Glover, F., Klingman, D.D., Phillips N.V. and Schneider, R.F. (1985b) New polynomial shortest path algorithms and their computational attributes. *Management Science*, **31**(9), 1106-1128.

Golden, B.L. and Magnanti, T.L. (1977) Deterministic network optimization: a bibliography. *Networks*, 7, 149-183.

Gopalan, R., Batta, R. and Karwan, M.H. (1990) The equity constrained shortest path problem. *Computers & Operations Research*, **17**(3), 297-307.

Goto, S. and Sangiovanni-Vincentelli, A. (1978) A new shortest path updating algorithm. *Networks*, **8**, 341-372.

Graham, R.L., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, **5**, 287–326.

Guinet, A. (1993) Scheduling sequence-dependent jobs on identical parallel machines to minimize completion criteria. *International Journal of Production Research*, **31**, 1579-1594.

Gupta, A.K. and Sivakumar, A.I. (2004) Single machine scheduling with multiple objectives in semiconductor manufacturing. *The International Journal of Advanced Manufacturing Technology*.

Gupta, D. and Magnusson, T. (2005) The capacitated lot-sizing and scheduling problem with sequence-dependent setup costs and setup times. *Computers & Operations Research*, **32**(4), 727-747.

Gupta, J.N.D. (1986) Flowshop schedules with sequence dependent setup times. *Journal of Operations Research Society of Japan*, **2**9, 206-219.

Gupta, J.N.D. and Darrow, W.P. (1986) The two-machine sequence dependent flowshop scheduling problem. *European Journal of Operational Research*, **24**, 439-446.

Gupta, S.K. (1982) *n* jobs and *m* machines job-shop problems with sequence-dependent setup times. *International Journal of Production Research*, **20**(5), 643-656.

Haase, K. (1994) Lotsizing and scheduling for production planning. *Lecture Notes in Economics and Mathematical Systems*, Springer-Verlag Publisher.

Haase, K. (1996) Capacitated lotsizing with sequence dependent setup costs. *Operations Research Spektrum*, **18**, 51-59.

Haase, K. and Kimms, A. (2000) Lot sizing and scheduling with sequence-dependent setup costs and times and efficient rescheduling opportunities. *International Journal of Production Economics*, **66**(2), 159-169.

Handler, G.Y. and Zang, I. (1980) A dual algorithm for the constrained shortest path problem. *Networks*, **10**, 293-310.

Hashiba, S. and Chang, T.C. (1991) PCB assembly set-up reduction using Group Technology. *Computers & Industrial Engineering*, **21**(1-4), 453-457.

Hassan, M.M.D. (1992) Network reduction for the acyclic constrained shortest path problem. *European Journal of Operational Research*, **63**, 124-132.

Hassin, R. (1992) Approximation schemas for the restricted shortest path problems. *Mathematics of Operations Research*, **17**(1), 36-42.

Heady, R.B. and Zhu, Z. (1998) Minimizing the sum of job earliness and tardiness in a multimachine system. *International Journal of Production Research*, **36**(6), 1619-1632.

Hertz, A. and Widmer, M. (1996) An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. *Discrete Applied Mathematics*, **65**, 319-345.

Holmberg, K. and Yuan, D. (2003) A multicommodity network-flow problem with side constraints on paths solved by column generation. *Informs Journal of Computing*, **15**(1), 42-57.

Hop, N.V. and Nagarur, N.N. (2004) The scheduling problem of PCBs for multiple non-identical parallel machines. *European Journal of Operational Research*, **158**, 577-594.

Houck, Jr., D.J., Picard, J.C., Queyranne, M. and Vemuganti, R.R. (1980) The traveling salesman problem as a constrained shortest path problem: Theory and Computational Experience. *Opsearch*, **17**, 93-109.

Hui, C., Gupta, A. and van der Meulen, H.A.J. (2000) A novel MILP formulation for short-term scheduling of multistage multi-product batch plants. *Computers & Chemical Engineering*, **24**(2-7), 1611-1617.

Hurink, J. and Knust, S. (2001) List scheduling in a parallel machine environment with precedence constraints and setup times. *Operations Research Letters*, **29**(5), 231-239.

Hwang, H. and Sun, J.U. (1997) Production sequencing problem with reentrant work flows and sequence dependent setup times. *Computers & Industrial Engineering*, **33**(3-4), 773-776.

Hwang, H. and Sun, J.U. (1998) Production sequencing problem with re-entrant work flows and sequence dependent setup times. *International Journal of Production Research*, **36**(9), 2435-2450.

Ioachim, I., Gélinas, S., Soumis, F. and Desrosiers, J. (1998) A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, **31**, 193-204.

Jaffe, J.M. (1984) Algorithms for finding paths with multiple constraints. *Networks*, **14**, 95-116.

Jaumard, B., Semet, F. and Vovor, T. (1996) A two-phase resource constrained shortest path algorithm for acyclic graphs. Les Cahiers du GERAD G-96-48, École des Hautes Études Commerciales, Montréal, Canada, H3T 2A7.

Jaumard, B., Semet, F. and Vovor, T. (1998) A generalized linear programming model for nurse scheduling. *European Journal of Operational Research*, **107**, 1-18.

Johnson, D.S. and Papadimitriou, C.H. (1985) Computational complexity. in *The Traveling Salesman Problem*, Lawler, E.L., Lenstra, J.K., Rinnooy, A.H.G. and Shmoys, D.B. (eds), Wiley, Chichester, UK, 37-85.

Joksch, H. C. (1966) The shortest route problem with constraints. *Journal of Mathematical Analysis and Applications*, **14**, 191-197.

Jordan, C. and Drexl, A. (1998) Discrete lotsizing and scheduling by batch sequencing. *Management Science*, **44**(5), 698-713.

Kang, S., Malik, K. and Thomas, L. (1999) Lotsizing and scheduling on parallel machines with sequence-dependent setup costs. *Management Science*, **45**(2), 273–289.

Karimi, B., Fatemi Ghomi, S.M.T. and Wilson, J.M. (2003) The capacitated lot-sizing problem: a review of models and algorithms. *Omega*, **31**, 365-378.

Katoh, N., Ibaraki, T. and Mine, H. (1982) An efficient algorithm for *k* shortest simple paths. *Networks*, **12**, 411-427.

Kim, C.O. and Shin, H.J. (2003) Scheduling jobs on parallel machines: a restricted tabu search approach. *The International Journal of Advanced Manufacturing Technology*, **22**(3-4), 278-287.

Kim, D., Kim, K., Jang, W. and Chen, F.F. (2002) Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, **18**(3-4), 223-231.

Kim, S.C. and Bobrowski, P.M. (1994) Impact of sequence-dependent setup time on job shop scheduling performance. *International Journal of Production Research*, **32**(7), 1503-1520.

Kim, S.C. and Bobrowski, P.M. (1997) Scheduling jobs with uncertain setup times and sequence dependency. *Omega*, **25**(4), 437-447.

Kim, S.-S., Shin, H.J., Eom, D.-H. and Kim, C.-O. (2003) A due date density-based categorising heuristic for parallel machines scheduling. *The International Journal of Advanced Manufacturing Technology*, **22**(9-10), 753-760.

Kim, S.-Y., Lee, Y.-H. and Agnihotri, D. (1995) A hybrid approach to sequencing jobs using heuristic rules and neural networks. *Production Planning & Control*, **6**(5), 445-454.

Kim, Y., Lim, H. and Park, M. (1996) Search heuristics for a flow shop scheduling problem in a printed circuit board assembly process. *European Journal of Operational Research*, **91**(1), 124-143.

King, V. and Thorup, M. (2001) A space saving trick for directed dynamic transitive closure and shortest path algorithms. in *Proceedings of the 7th Annual International Computing and Combinatorial Conference COCOON in LNCS 2108*, 268-277, Springer-Verlag.

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

Kolahan, F. and Liang, M. (1998) An adaptive TS approach to JIT sequencing with variable processing times and sequence-dependent setups. *European Journal of Operational Research*, **109**, 142-159.

Kolahan, F., Liang, M. and Zuo, M. (1995) Solving the combined part sequencing and tool replacement problem for an automated machine center: a tabu search approach. *Computers & Industrial Engineering*, **28**, 731-743.

Kolen, A., Rinnooy Kan, A. and Trienekens, H. (1987) Vehicle routing with time windows. *Operations Research*, **35**, 266-273.

Kurz, M.E. and Askin, R.G. (2001) Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, **39**(16), 3747-3769.

Kurz, M.E. and Askin, R.G. (2003) Comparing scheduling rules for flexible flow lines. *International Journal of Production Economics*, **85**(3), 371-388.

Kurz, M.E., and Askin, R.G. (2004) Scheduling flexible flow lines with sequence-dependent setup times. *European Journal of Operational Research*, **159**, 66-82.

Laguna, M. (1999) A heuristic for production scheduling and inventory control in the presence of sequence-dependent setup times. *IIE Transactions*, **31**(2), 125-134.

Laguna, M. and Glover, F.W. (1993) Integrating target analysis and tabu search for improved scheduling systems. *Expert Systems with Applications*, **6**, 287-297.

Lavoie, S., Minoux, M. and Odier, E. (1988) A new approach for crew pairing problems by column generation with application to air transportation. *European Journal of Operational Research*, **35**, 45-48.

Lawler, E.L. (1976) *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart &Winston, New York, NY.

Lee, Y.H., Bhaskaran, K. and Pinedo, M. (1992) A heuristic to minimize the total weighted tardiness with sequence dependent setups. *Technical Report*, IEOR Dept., Columbia University, New York.

Lee, Y.H., Bhaskaran, K. and Pinedo, M. (1997) A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, **29**(1), 45-52.

Lee, Y.H. and Pinedo, M. (1997) Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, **100**, 464–474.

Leon, V.J. and Wu, S.D. (1992) On scheduling with ready-times, due-dates and vacations. *Naval Research Logistics*, **39**, 53-65.

Little, J.D.C., Murty, K.G., Sweeney, D.W. and Karel, C. (1963) An algorithm for the traveling salesman problem. *Operations Research*, **11**, 972-989.

Logendran, R., Gelogullari, C.A. and Sriskandarajah, C. (2003) Minimizing the mean flow time in a two-machine group-scheduling problem with carryover sequence dependency. *Robotics and Computer Integrated Manufacturing*, **19**, 21-33.

Lorenz, D.H. and Raz, D. (2001) A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letter*, **28**, 213-219.

Low, C. (1995) Job shop scheduling heuristics for sequence dependent setups. *Computers & Industrial Engineering*, **29**(1-4), 279-283.

Luh, P.B., Gou, L., Zhang, Y., Nagahora, T., Tsuji, M., Yoneda, K., Hasegawa, T., Kyoya, Y. and Kano T. (1998) Job shop scheduling with group-dependent setups, finite buffers and long time horizon. *Annals of Operations Research: Mathematics of Industrial Systems*, **76**, 233-259.

Maimon, O.Z., Dar-El, E.M. and Carmon, T.F. (1993) Set-up saving schemes for printed circuit boards assembly. *European Journal of Operational Research*, **70**, 177-190.

Mason, S.J., Fowler, J.W. and Matthew W.C. (2002) A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling*, **5**(3), 247-262.

Maxwell, W.L. (1964) The scheduling of economic lot sizes. *Naval Research Logistics Quarterly*, **11**(2-3), 89-124.

McGinnis, L.F., Ammons, J.C., Carlyle, M., Cranmer, L., Depuy, G.W., Ellis, K.P., Tovey, C.A. and Xu, H. (1992) Automated process planning for printed circuit card assembly. *IIE Transactions*, **24**(4), 18-30.

Mehlhorn, K. and Ziegelmann, M. (2000) Resource Constrained shortest paths. in *7th Ann European Symp on Algorithms* (ESA 2000), Paterson, M. (ed), LNCS 1879, Springer-Verlag, Heidelberg, Berlin, 326-337.

Mendes, A.S., Müller, F.M., França, P.M. and Moscato, P. (2002) Comparing meta-heuristic approaches for parallel machine scheduling problems. *Production Planning & Control*, **13**(2), 143-154.

Mendez, C.A., Henning, G.P. and Cerde, J. (2001) An MILP continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities. *Computers & Chemical Engineering*, **25**(4-6), 701-711.

Meyr, H. (2000) Simultaneous lotsizing and scheduling by combining local search with dual reoptimization. *European Journal of Operational Research*, **120**(2), 311-326.

Meyr, H. (2002) Simultaneous lotsizing and scheduling on parallel machines. *European Journal of Operational Research*, **139**(2), 277-292.

Miller, M.D., Chen, H.-C., Matson, J. and Liu, Q. (1999) A hybrid genetic algorithm for the single machine scheduling problem. *Journal of Heuristics*, **5**, 437-454.

Mingozzi, A., Boschetti, M.A., Ricciardelli, S. and Bianco, L. (1999) A set partitioning approach to the crew scheduling problem. *Operations Research*, **47**(6), 873-888.

Minoux, M. and Ribeiro, C. (1984) A transportation of hard (Equality constrained knapsack problems into constrained shortest path problems. *Operations Research Letters*, **3**, 211-214.

Mokotoff, E. (2001) Parallel machine scheduling problem: a survey. *Asia-pacific Journal of Operational Research*, **18**, 193-242.

Monma, C.L. and Potts, C.N. (1989) On the complexity of scheduling with batch setup times. *Operations Research*, **37**(5), 798-804.

Moscato, P. (1989) On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. *Caltech Concurrent Computation Program, C3P Report 826*.

Moscato, P. (1999) Memetic algorithms: a short introduction. in *New Ideas in Optimization*, Corne, D., Dorigo, M. and Glover F. (eds), McGraw-Hill, London, 219-234.

Nauss, R.M. (1978) The 0-1 knapsack problem with multiple choice constraint. *European Journal of Operational Research*, **2**, 125-131.

Norman, B.A. (1999) Scheduling flowshops with finite buffers and sequence-dependent setup times. *Computers & Industrial Engineering*, **36**(1), 163-177.

OR-library, RCSP instances, http://people.brunel.ac.uk/~mastjjb/jeb/orlib/rcspinfo.html. Accessed April, 2005.

Osman, I.H. and Potts, C.N. (1989) Simulated annealing for permutation flow-shop scheduling. *Omega*, **17**(6), 551–557.

Ovacik, I.M. and Uzsoy, R. (1993) Worst-case error bounds for parallel machines scheduling problems with sequence-dependent setup times. *Operations Research Letters*, **14**, 251-256.

Ovacik, I.M. and Uzsoy, R. (1994a) Exploiting shop floor status information to schedule complex job shops. *Journal of Manufacturing Systems*, **13**(2), 73-84.

Ovacik, I.M. and Uzsoy, R. (1994b) Rolling horizon algorithms for a single-machine dynamic scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, **32**(6), 1243-1263.

Ovacik, I.M. and Uzsoy, R. (1995) Rolling horizon procedures for dynamic parallel machine scheduling with sequence-dependent setup times. *International Journal of Production Research*, **33**(11), 3173-3192.

Ovacik, I.M. and Uzsoy, R. (1997) *Decomposition Methods for Complex Factory Scheduling Problems*, Kluwer Academic Publishers, Dordrecht, MA.

Ozgur, C.O. and Brown, J.R. (1995) A two-stage traveling salesman procedure for the single machine sequence-dependent scheduling problem. *Omega*, **23**(2), 205-219.

Pallottino, S. and Scutellà, M.G. (2003) A new algorithm for reoptimizing shortest paths when the arc costs change. *Operations Research Letters*, **31**, 149-160.

Palmer, D.S. (1965) Sequencing jobs through a multistage process in the minimum total time - a quick method of obtaining a near optimum. *Operational Research Quarterly*, **16**(1), 101-107.

Papadimitriou, C.H. and Steiglitz, K. (1998) *Combinatorial Optimization: Algorithms and Complexity*, Dover, Mineola, NY.

Park, Y., Kim, S. and Lee, Y. (2000) Scheduling jobs on parallel machines applying neural network and heuristic rules. *Computers & Industrial Engineering*, **38**(1), 189-202.

Parthasarathy, S. and Rajendran, C. (1997a) An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times of jobs. *International Journal of Production Economics*, **49**(3), 255-263.

Parthasarathy, S. and Rajendran, C. (1997b) A simulated annealing heuristic for scheduling to minimize mean weighted tardiness in a flowshop with sequence-dependent setup times of jobs-a case study. *Production Planning & Control*, **8**(5), 475-483.

Pearn, W.L., Chung, S.H., Chen, A.Y. and Yang, M.H. (2004) A case study on the multistage IC final testing scheduling problem with reentry. *International Journal of Production Economics*, **88**(3), 257-267.

Pinedo, M. (2002) *Scheduling: Theory, Algorithms and Systems*, 2nd edn., Prentice Hall, Englewood Cliffs, NJ.

Pisinger, D. (1994) A minimal algorithm for the multiple-choice knapsack problem. *Technical Report* 94/25, DIKU, University of Copenhagen, Denmark.

Potts, C.N. and van Wassenhove, L.N. (1992) Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of Operations Research Society*, **43**(5), 395-406.

Pugazhendhi, S., Thiagarajan, S., Rajendran, C. and Anantharaman, N. (2004) Generating non-permutation schedules in flowline-based manufacturing systems with sequence-dependent setup times of jobs: a heuristic approach. *The International Journal of Advanced Manufacturing Technology*, **23**(1-2), 64-78.

Rabadi, G., Mollaghasemi, M. and Anagnostopoulos, G.C. (2004) A branch-and-bound algorithm for the early/tardy machine scheduling problem with a common due-date and sequence-dependent setup time. *Computers & Operations Research*, **31**, 1727-1751.

Radhakrishnan, S. and Ventura, J.A. (2000) Simulated Annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. *International Journal of Production Research*, **38**(10), 2233-2252.

Ragatz, G.L. (1993) A branch-and-bound method for minimum tardiness sequencing on a single processor with sequence dependent setup times, in *Proceedings: twenty-fourth annual meeting of the Decision Sciences Institute*, **23**(4), pp. 346-354.

Rajendran, C. and Ziegler, H. (1997) A heuristic for scheduling to minimize the sum of weighted flowtime of jobs in a flowshop with sequence-dependent setup times of jobs. *Computers & Industrial Engineering*, **33**(1-2), 281-284.

Rajendran, C. and Ziegler, H. (2003) Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. *European Journal of Operational Research*, **149**(3), 513-522.

Rajkumar, K. and Narendran, T.T. (1998) A heuristic for sequencing PCB assembly to minimize setup times. *Production Planning & Control*, **9**(5), 465-476.

Ramalingam, G. and Reps, T. (1996) An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, **21**, 267-305.

Raman, N., Rachamadugu, R.V. and Talbot, F.B. (1989) Real time scheduling of an automated manufacturing center. *European Journal of Operational Research*, **40**, 222-242.

Reeves, C.R. (1995) A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, **22**(1), 5–13.

Reinelt, G. (1994) *The Traveling Salesman: Combinatorial Solutions for TSP Applications*, Springer-Verlag, Berlin.

Ribeiro C.C. and Minoux M. (1985) A heuristic approach to hard constrained shortest path problems. *Discrete Applied Mathematics*, **10**, 125-137.

Rios-Mercado, R.Z. and Bard, J.F. (1998a) Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. *Computers & Operations Research*, **25**(5), 351-366.

Rios-Mercado, R.Z. and Bard, J.F. (1998b) Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, **110**(1), 76-98.

Rios-Mercado, R.Z., and Bard, J.F. (1999a) A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. *IIE Transactions*, **31**(8), 721-731.

Rios-Mercado, R.Z., and Bard, J.F. (1999b) An enhanced TSPbased heuristic for makespan minimization in a flow shop with setup times. *Journal of Heuristics*, **5**(1), 57-74.

Roslöf, J., Harjunkoski, I., Westerlund, T. and Isaksson J. (2002) Solving a large-scale industrial scheduling problem using MILP combined with a heuristic procedure. *European Journal of Operational Research*, **138**, 29-42.

Rossetti, M.D. and Stanford, K.J.A. (2003) Group sequencing a PCB assembly system via an expected sequence dependent setup heuristic. *Computers & Industrial Engineering*, **45**(1), 231-254.

Roy, B. and Sussman, B. (1964) Les problèmes d'ordonnancement avec contraintes disjonctives, in *Proceedings of SEMA*, Montrouge, France.

Rubin, P.A. and Ragatz, G.L. (1995) Scheduling in a sequence dependent setup environment with genetic search. *Computers & Operations Research*, **22**(1), 85-99.

Ruiz, R., Maroto, C. and Alcaraz, J. (2005) Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research*, **165**, 34-54.

Russell, R.A. (1995) Hybrid heuristics for the vehicle-routing problem with time windows. *Transportation Sciences*, **29**, 156-166.

Salomon, M., Solomon, L., van Wassenhove, L.N., Dumas, J. and Dauzere-Peres, S. (1997) Solving the discrete lotsizing and scheduling problem with sequence dependent set-up costs and set-up times using the Traveling Salesman Problem with time windows. *European Journal of Operational Research*, **100**, 494-513.

Schaller, J.E., Gupta, J.N.D. and Vakharia, A.J. (2000) Scheduling a flowline manufacturing cell with sequence dependent family setup times. *European Journal of Operational Research*, **125**(2), 324-339.

Schutten, J.M.J. (1996) List scheduling revisited. *Operations Research Letters*, **18**(4), 167-170.

Shailendra, J., Eric, J.M. and Fereydoon, S. (1996) Implementing setup optimization on the shop floor. *Operations Research*, **44**(6), 843-851.

Sherali, H.D., Hobeika, A.G., and Kangwalklai, S. (2003) Time-dependent, label-constrained shortest path problems with applications. *Transportation Science*, **37**(3), 278-293.

Shin, W.S., Hart, S.M. and Lee, H.F. (1995) Strategic allocation of inspection stations for a flow assembly line: a hybrid procedure. *IIE Transactions*, **27**, 707-715.

Sikora, R. (1996) A genetic algorithm for integrating lot-sizing and sequencing in scheduling a capacitated flow line. *Computers & Industrial Engineering*, **30**(4), 969-981.

Sikora, R., Chhajed, D. and Shaw, M.J. (1996) Integrating the lot-sizing and sequencing decisions for scheduling a capacitated flow line. *Computers & Industrial Engineering*, **30**(4), 659-679.

Silver, E.A. and Meal. H.C. (1973) A heuristic selecting lot size requirement for the case of a deterministic time varying demand rate and discrete opportunities for replenishment. *Production and Inventory Management*, **14**, 64-74.

Simons, Jr. J.V. (1992) Heuristics in flow shop scheduling with sequence dependent setup times. *Omega*, **20**(2), 215-225.

Sinha, A. and Zoltners, A.A. (1979) The multiple-choice knapsack problem. *Operations Research*, **27**, 503-515.

Sivrikaya-Serifoglu, F. and Ulusoy, G. (1999) Parallel machine scheduling with earliness and tardiness penalties. *Computers & Operations Research*, **26**(8), 773-787.

Spina, R., Galantucci, L.M. and Dassisti, M. (2003) A hybrid approach to the single line scheduling problem with multiple products and sequence-dependent time. *Computers & Industrial Engineering*, **45**(4), 573-583.

Srikar, B.N. and Ghosh, S. (1986) A MILP model for the *n*-job, *M*-stage flowshop, with sequence dependent setup times. *International Journal of Production Research*, **24**(6), 1459-1472.

Stafford, Jr. E.F. and Tseng, F.T. (1990) On the Sriker-Ghosh MILP model for the *N*×*M* SDST flowshop problem. *International Journal of Production Research*, **28**(10), 1817-1830.

Stafford, Jr. E.F. and Tseng, F.T. (2002) Two models for a family of flowshop sequencing problems. *European Journal of Operational Research*, **142**(2), 282-293.

Stojković, M. and Soumis, F. (2001) An Optimization Model for the Simultaneous Operational Flight and Pilot Scheduling Problem. *Management Science*, **47**(9), 1290-1305.

Stutzle, T. (1998) Applying iterated local search to the permutation flow shop problem. *Technical Report*, AIDA-98-04, Darmstadt, Germany.

Sun, J.U. and Hwang, H. (2001) Scheduling problem in a two-machine flow line with the *N*-step prior-job-dependent set-up times. *International Journal of Systems Science*, **32**(3), 375-385.

Sun, J.U., Yee, S.R. and Hwang, H. (2003) Job shop scheduling with sequence dependent setup times to minimize makespan. *International Journal of Industrial Engineering – Theory Applications and Practice*, **10**(4), 455-461.

Sun, X., Noble, J.S. and Klein, C.M. (1999) Single-machine scheduling with sequence dependent setup to minimize total weighted squared tardiness. *IIE Transactions*, **31**(2), 113-124.

Taillard, E. (1993) Benchmarks for basic scheduling problems. *European Journal of Operational Research*, **64**, 278–285.

Tan, K.C. and Narasimhan, R. (1997a) Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach. *Omega*, **25**(6), 619-634.

Tan, K.C. and Narasimhan, R. (1997b) Multi-objective sequencing with sequence dependent setup times. *International Journal of Operations and Quantitative Management*, **3**, 69-84.

Tan, K.C., Narasimhan, R., Rubin, P.A. and Ragatz, G.L. (2000) A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. *Omega*, **28**(3), 313-326.

Thizy, J.M. and van Wassenhove, L.N. (1985) Lagrangean relaxation for the multi-item capacitated lot-sizing problem: a heuristic implementation. *IEE Transactions*, **17**(4), 308-313.

Trigeiro, W.W., Thomas, L.J. and McClain, J.O. (1989) Capacitated lot sizing with setup times. *Management Science*, **35**(3), 353-366.

Unal, A.T., Uzsoy, R. and Kiran, A.S. (1997) Rescheduling on a single machine with part-type dependent setup times and deadlines. *Annuals of Operations Research*, **70**, 93-113.

Uzsoy, R., Lee, C.Y. and Martin-Vega, L.A. (1992) Scheduling semiconductor test operations: minimizing maximum lateness and number of tardy jobs on a single machine. *Naval Research Logistics*, **39**, 369-388.

Uzsoy, R., Martin-Vega, L.A., Lee, C.Y. and Leonard, P.A. (1991) Production scheduling algorithms for a semiconductor testing facility. *IEEE Transactions on Semiconductor Manufacturing*, **4**, 270-280.

Vallada, E., Ruiz, R. and Maroto, C. (2003) Synthetic and real benchmarks for complex flow-shop problems. *Technical Report*, Universidad Politécnica de Valencia, Valencia, España, Grupo de Investigatión Operativa GIO.

van den Akker, J.M., Hurkens, C.A.J. and Savelsbergh, M.W.P. (2000) A time-indexed formulation for single-machine scheduling problems: column generation. *INFORMS Journal on Computing*, **12**(2), 111-124.

Villeneuve, D. and Desaulniers, G. (2005) The shortest path problem with forbidden paths. *European Journal of Operational Research*, **165**(1), 97-107.

Wagner, B.J. and Davis, D.J. (2002) A search heuristic for the sequence-dependent economic lot scheduling problem. *European Journal of Operational Research*, **141**(1), 133-146.

Weng, M.X., Lu, J. and Ren, H. (2001) Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International Journal of Production Economics*, **70**(3), 215-226.

White, L.S. (1969) Shortest route models for the allocation of inspection effort on a production line. *Management Science*, **15**, 249-259.

Widmer, M. and Hertz, A. (1989) A New heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, **41**, 186-193.

Wilbrecht, J.K. and Prescott, W.B. (1969) The influence of set-up time on job shop performance. *Management Science*, **16**, B274-B280.

Wilhelm, W.E. (1999) A column-generation approach for the assembly system design problem with tool changes. *The International Journal of Flexible Manufacturing Systems*, **11**, 177-205.

Wilhelm, W.E. (2001) A technical review of column generation in integer programming. *Optimization and Engineering*, **2**(2), 159-200.

Wilhelm, W.E., Arambula, I. and Choudhry, N.N. (2005a) A model to optimize picking operations on dual-head placement machines. *IEEE Transactions on Automation Science and Engineering*, forthcoming.

Wilhelm, W.E., Choudhry, N.D. and Damodaran, P. (2004) A model to optimize placement operations on dual-head placement machines. In review.

Wilhelm, W.E., Damodaran, P. and Li, J. (2003) Prescribing the content and timing of product upgrades. *IIE Transactions*, **35** (7), 647-664.

Wilhelm, W.E., Liang, D., Bulusu, S., Warrier, D., Zhu, X. and Vasudeva, B.R.T. (2005b) Design of international assembly systems and their supply chains under NAFTA. *Transportation Research E*, in press.

Wilhelm, W.E. and Tarmy P.K. (2003) Circuit card assembly on tandem turret-type placement machines. *IIE Transactions*, **35**, 627-645.

Woodruff, D.L. and Spearman, M.L. (1992) Sequencing and batching for two classes of job with deadlines and setup times. *Production and Operations Management*, **1**, 87-102.

Yalaoui, F. and Chu, C. (2003) An efficient heuristic approach for parallel machine scheduling with job splitting and sequence-dependent setup times. *IIE Transactions*, **35**, 183-190.

Yang, W.H. and Liao, C.J. (1999) Survey of scheduling research involving setup times. *International Journal of system Science*, **30**(2), 143-155.

Yen, J.Y. (1971) Finding the *k* shortest loopless paths in a network. *Management Science*, **17**, 712-716.

Zoghby, J., Barnes, J.W. and Hasenbein, J.J. (2005) Modeling the reentrant job shop scheduling problem with setups for metaheuristic searches. *European Journal of Operational Research*, **167**, 336-348.

VITA

The author of this dissertation, Xiaoyan Zhu, graduated from Tsinghua University, Beijing, China, in June 2000 with a Bachelor of Engineering degree in thermal power engineering. Her outstanding academic performance made it possible for her to join the Singapore-Massachusetts Institute of Technology Alliance (*SMA*), which is provided by Massachusetts Institute of Technology, Boston, and Nanyang Technological University, Singapore, to pursue her first Master of Science degree in innovation in manufacturing systems and technology. She graduated from this program in June 2001.

She received her second Master of Science degree from Texas A&M University, College Station, in industrial engineering in December 2002. She received her Doctor of Philosophy degree from Texas A&M University, College Station, in industrial engineering in December 2005.

Her permanent address is #1 Yangluo Main Street, XinZhou District, Wuhan City, Hubei Province 430415, China.

The typist for this dissertation was Xiaoyan Zhu.