

A Branch-and-Price Algorithm for Parallel Machine Scheduling with Time Windows and Job Priorities

Jonathan F. Bard,¹ Siwate Rojanasoonthon²

¹Graduate Program in Operations Research and Industrial Engineering, 1 University Station, C2200, University of Texas, Austin, Texas 78712

²TMB Asset Management Co. Ltd., 990 Rama IV Road, Silom, Bangrak, Bangkok 10500, Thailand

Received 22 November 2004; revised 9 August 2005; accepted 20 August 2005

DOI 10.1002/nav.20118

Published online 10 November 2005 in Wiley InterScience (www.interscience.wiley.com).

Abstract: This paper presents a branch-and-price algorithm for scheduling n jobs on m nonhomogeneous parallel machines with multiple time windows. An additional feature of the problem is that each job falls into one of ρ priority classes and may require two operations. The objective is to maximize the weighted number of jobs scheduled, where a job in a higher priority class has “infinitely” more weight or value than a job in a lower priority class. The methodology makes use of a greedy randomized adaptive search procedure (GRASP) to find feasible solutions during implicit enumeration and a two-cycle elimination heuristic when solving the pricing subproblems. Extensive computational results are presented based on data from an application involving the use of communications relay satellites. Many 100-job instances that were believed to be beyond the capability of exact methods, were solved within minutes. © 2005 Wiley Periodicals, Inc. *Naval Research Logistics* 53: 24–44, 2006.

Keywords: parallel machine scheduling; time windows; branch and price; GRASP; job priorities

1. INTRODUCTION

Machine scheduling problems are characterized by sequencing and timing decisions and arise in situations where activities compete for scarce resources. Depending on the environment (e.g., single machine or parallel machines), the job characteristics (e.g., independent or precedence constrained), and optimality criteria (e.g., makespan, total tardiness), it is possible to define a wide range of problem subclasses.

In this paper, we address a parallel machine scheduling problem in which each job requires either one or two operations that can be performed on one or more parallel machines. Once the operations are completed, the machine becomes free and the job leaves the system. The three common classes of parallel machine environments include identical machines, uniform machines, and unrelated machines. In the first environment, a job can be processed in the same amount of time on any of the machines. In the second, each machine may process a particular job at different speed, so a job would require less processing time on a fast machine compared with a slower one. In general, each machine has an associated speed factor. In the third environment, the processing time of jobs can be

completely arbitrary. One machine might be faster on one job but slower on another (Cheng and Sin [23]).

Our problem falls into the third category and originated from a real-world need to schedule the use of antennas on the Tracking and Data Relay Satellite System (TDRSS) (Rojanasoonthon, Bard, and Reddy [21]). In this system, satellites (machines) in geosynchronous orbit are used as relay platforms for communication between low orbiting spacecraft and ground terminals. Spacecraft can communicate with the TDRSS only when they are within line-of-sight range. Users (spacecraft), with unique priorities, request blocks of contiguous antenna time by specifying the antenna preference, the contacting priority, and the time period in which contact is possible (time window). Each of the six TDRSS antennas can handle at most one request (job) at a time. In some situations, users may request periodic contacts or multiple simultaneous contacts. The objective is to allocate the antennas over the planning period to meet the users' requests in such a way that some function of the scheduled requests is maximized with strict priority enforcement. This means that a subset of jobs from one priority class can be scheduled independently of all other jobs with lower priorities. In other words, scheduling a job in higher priority class benefits infinitely more than scheduling any number of lower priority jobs.

Correspondence to: J.F. Bard (jbard@mail.utexas.edu)

The purpose of this paper is to show how Dantzig–Wolfe decomposition can be combined with intelligent heuristics to solve extremely difficult parallel machine scheduling problems with time windows and job priorities. Up until now, instances with only a handful of jobs could be solved to optimality with exact methods. The proposed branch-and-price algorithm provides an order of magnitude improvement over current techniques.

In the TDRSS application, the planning period spans 24 h, the number of priority classes is about 20, and the number of requests may be as high as 400 per day. The orbital system consists of two satellites, each having two single access antennas and one phased array multiple access antenna (i.e., 3 distinct machines). Because the time windows of a job are a function of the relative positions of the user spacecraft and the satellites, the antennas can be viewed as six distinct machines, although two on each satellite are identical. Processing times vary from a few seconds to 10 min while the time windows can range up to several hours. In some cases, the processing time of a job spans its entire time window. For those jobs that can be processed by either type of antenna, the phased array is always faster.

In the next section, we give a general statement of the problem, followed in Section 3 by an outline of the relevant literature. In Section 4, the mathematical formulation of the problem is presented. Several unsuccessful attempts to solve the corresponding mixed-integer linear program with a commercial code led to the development of the branch-and-price algorithm detailed in Section 5. Test results are highlighted in Section 6, where it is seen that instances ranging in size from 20 jobs and six machines up to 100 jobs and two machines can be solved in less than 10 min in almost all cases.

2. PROBLEM DEFINITION

We wish to schedule n jobs on m nonhomogeneous parallel machines without preemption to maximize the total weighted sum of processed jobs. Each job falls into one of ρ priority classes and must be processed within its specified time window. An optimal schedule for this version of the parallel machine scheduling problem with time windows (PMSPTW) is one that satisfies the following conditions:

1. Each job is processed within its time window; otherwise, it is considered unscheduled. (In some cases, a job may have two time windows and so may be processed within either.)
2. If a job is selected, then it is processed exactly once without interruption on one of its permissible machines if it requires only one service (operation); if it requires two services, two different machines may be used, time windows permitting.

3. Each machine processes at most one job at a time and may require a sequence-dependent setup between jobs.
4. The total contribution of the scheduled jobs is maximized.

Using the standard three-field notation (Graham et al. [14]), PMSPTW may be written as $Rm|r_j, \text{deadline}, M_j, s_{jk}|\sum w_j U_j$, where Rm denotes unrelated parallel machines, r_j is the release date, *deadline* implies a hard due-date constraint for each job, M_j represents the machine-eligibility restrictions, s_{jk} is the sequence-dependent setup time, and U_j is one if job j is processed and 0 otherwise. The objective is to maximize the weighted number of scheduled jobs. The weights, w_j , must be assigned so that the optimal solution reflects the strict enforcement of priorities.

The PMSPTW possesses many of the characteristics of the multiple traveling salesman problem with time windows (m -TSPTW) which, itself, is a special case of the vehicle routing problem (VRP) with time windows (e.g., see Bard, Kontoravdis, and Yu [3], Desrochers, Desrosiers, and Solomon [8]). When solving VRPs, one of several objectives may be specified: Minimize the total routing cost, minimize the number of vehicles required, or minimize the distanced traveled. The objective of the problem addressed here is to maximize a function of customers visited (or minimize a function of unvisited customers). In generic terms, the VRP is stated as a covering problem while the PMSPTW is stated as a packing problem.

3. LITERATURE REVIEW

Among the many survey papers on machine scheduling, Blazewicz, Dror and Weglarz [24] primarily focused on mathematical formulations. They covered single machine scheduling with several different objective functions and a variety of operational constraints. For parallel machine problems, they discussed formulations for both nonpreemptive and preemptive scheduling of identical, uniform, and unrelated machines. Lawler et al. [18] provided an overview of sequencing and scheduling problems and related solution techniques.

The literature on parallel machine scheduling has expanded rapidly in recent years largely due to the development of intelligent heuristics. Our review concentrates on the use of exact methods to solve problems that are closely related to the PMSPTW. For a full survey, see Rojanasoonthon [20].

3.1. Scheduling Jobs with Fixed Start and End Times on Parallel Machines

Arkin and Silverberg [1] addressed a parallel machine scheduling problem where each job i had a positive weight,

a fixed start time, a fixed end time (s_i and t_i , respectively), and corresponding processing times $t_i - s_i$. The objective was to maximize the total value of the scheduled jobs. Preemption was prohibited and jobs could be processed at most once. This problem is similar, but not identical, to ours. For the case where all the machines are identical, an $O(n^2 \log n)$ algorithm was presented based on formulating the problem as a minimum-cost flow network.

Kolen and Kroon [16] dealt with a scheduling problem where each job had a fixed start time, a fixed end time, and a value representing its priority. Machines were available in specific time intervals (shifts), and a job could be processed only if the interval between the start and end time was a subinterval of a machine's shift. At most one job could be processed at a time and preemptions were not allowed. The objective was to find a feasible schedule for all jobs when such a schedule existed. Otherwise, the objective was to find a feasible schedule in which the subset of processed jobs yielded the maximum total value.

Bouzina and Emmons [5] studied interval scheduling where jobs with fixed start and end times were processed on identical parallel machines. The objective was to find a feasible schedule that maximized the number of completed jobs. When job weights were defined, the problem was to find a solution that maximized the sum of the weights of completed jobs. The unweighted and weighted versions were referred to as the *maximal IS* and *maximal-weight IS*, respectively. For maximal IS, an $O(n \max\{\log n, m\})$ algorithm was presented. For maximal-weight IS, the n -job problem was reformulated as a minimal-cost flow problem with $n + 1$ nodes and $2n$ arcs.

3.2. Scheduling Jobs with Time Windows and Job Priorities on Parallel Machines

Gabrel [12] focused on the problem of scheduling n nonpreemptive jobs, each with a given processing time and an interval for the start time, on m identical parallel machines. An additional constraint limited each job to be processed only on a subset of machines. Two variations of the problem were considered: (1) The fixed job scheduling problem where the interval of each job's start time was a point, and (2) the variable job scheduling problem where the interval was a nonzero range. The majority of the work dealt with developing upper and lower bounds for the fixed job case.

3.3. Orienteering Problem

This problem is defined by a starting point, a terminus, and a number of intermediate locations having associated scores. Also included is a fixed planning horizon, which implies that it may be possible to visit only a subset of the locations. The objective is to select a feasible subset of locations such that the total score collected achieves its maximum value.

The orienteering problem is a generalization of the traveling salesman problem and sometimes is called the traveling salesman's subtour problem, the maximum collection problem, or the selective traveling salesman problem (e.g., see Butt and Cavalier [6], Laporte and Martello [17]). Applications include finding a route for a specialist who can provide support to only a restricted number of customers (preferring those with higher sales potential) due to his availability constraint, routing of oil tankers to service ships at various locations, and scheduling the daily operation of a steel rolling mill (Balas [2], Golden, Levy, and Vohra [13]). The PMSPTW can be viewed of as a special case—a multiple tour maximum collection orienteering problem with time windows.

4. MATHEMATICAL FORMULATION

In this section, we define notation and present a constraint-based formulation for the PMSPTW when each job has one service. The model is then extended to the case where a job may require two services. To begin, let n be the number of jobs and m the number of machines. Each job in the set $J = \{1, \dots, n\}$ is a candidate to be processed without interruption on a set of parallel, nonhomogeneous machines $M = \{1, \dots, m\}$. Associated with each job is a time window $[a_{jk}, b_{jk}]$ (for job j on machine k , the earliest time processing of j can begin on machine k is a_{jk} and the latest time processing can begin is b_{jk} ; $0 \leq a_{jk} \leq b_{jk}$); a processing time p_{jk} (the processing time of job j on machine k); and a set of machines on which job j can be processed $M_j \subseteq M$. Between jobs i and j there is a setup time, s_{ijk} , if job i immediately precedes job j on machine k .

Also, there are ρ priority classes such that $J_p \subseteq J$, $p \in \{1, \dots, \rho\}$ is the set of jobs with priority p . If π_p is the contribution of a priority p job and β_i is the relative benefit derived from processing job i with respect to the other jobs in its priority class, then the total contribution of priority p jobs is $\pi_p \sum_{i \in J_p} \beta_i$. The strict enforcement of priorities requires $\pi_p > \sum_{q=p+1}^{\rho} \pi_q \sum_{i \in J_q} \beta_i$, $p = 1, \dots, \rho - 1$. Note that some job j may have two time windows, (a_{jk}^1, b_{jk}^1) and (a_{jk}^2, b_{jk}^2) , and may be processed in either one of them but not both.

In the developments, we make use of the following additional notation.

Indices and sets:

$J^1(J^2)$ = job sets with one (two) time windows, respectively.

F_{ij} = set of machines on which both jobs i and j can be processed; $F_{ij} = M_i \cap M_j$, $i, j \in J$.

0 = dummy job that represents the starting and ending of the sequences on each machine, $J^0 = J \cup 0$.

Parameters:

P_i = priority class of job $i \in J$.
 n_p = total number of jobs with priority p ($n_p = |J_p|$).
 β_i = relative benefit associated with processing job $i \in J$ with respect to the other jobs in P_i .
 π_p = contribution to the objective function of a priority p job, where class 1 is the highest class and ρ is the lowest; $\pi_p > \sum_{q=p+1}^{\rho} \pi_q \sum_{i \in J_q} \beta_i$, $p = 1, \dots, \rho - 1$, and $\pi_\rho = 1$.
 μ_{ij} = large number; $\mu_{ij} = \max_{k \in M_i} (b_{ik}) - \min_{k \in M_j} (a_{jk})$, $i, j \in J$; $i \neq j$.

Variables:

Flow variables: x_{ij}^k ($i, j \in J^0 : i \neq j, k \in F_{ij}$), where $x_{ij}^k = 1$ if machine k processes job j immediately after job i and $x_{ij}^k = 0$ otherwise; note that $x_{ij}^k = 0$ for all the undefined indices.

Time variables: t_j ($j \in J$) is the start time of job j .

Time window variables: y_{ik}^1 and y_{ik}^2 ($i \in J^2, k \in M_i$), where $y_{ik}^1 = 1$ if job i is processed in its first time window on machine k and $y_{ik}^1 = 0$ otherwise; $y_{ik}^2 = 1$ if job i is processed in its second time window on machine k and $y_{ik}^2 = 0$ otherwise.

Model 1: One Service Only for Each Job

$$\text{Maximize} \quad \sum_{i \in J} \pi_{P_i} \beta_i \left(\sum_{j \in J^0 \setminus \{i\}} \sum_{k \in M} x_{ij}^k \right) \quad (1)$$

subject to

$$\sum_{j \in J^0 \setminus \{i\}} \sum_{k \in M} x_{ij}^k \leq 1, \quad i \in J \quad (2)$$

$$\sum_{j \in J} x_{0j}^k \leq 1, \quad k \in M \quad (3)$$

$$\sum_{i \in J^0 \setminus \{j\}} x_{ij}^k - \sum_{i \in J^0 \setminus \{j\}} x_{ji}^k = 0, \quad j \in J^0, k \in M \quad (4)$$

$$t_i + \sum_{k \in M} (p_{ik} + s_{ijk}) x_{ij}^k - \left(1 - \sum_{k \in M} x_{ij}^k \right) \mu_{ij} \leq t_j, \quad i, j \in J, i \neq j \quad (5)$$

$$y_{ik}^1 + y_{ik}^2 - \sum_{j \in J^0 \setminus \{i\}} x_{ij}^k = 0, \quad i \in J^2, k \in M \quad (6)$$

$$\sum_{k \in M_i} a_{ik} \sum_{j \in J^0 \setminus \{i\}} x_{ij}^k \leq t_i, \quad i \in J^1 \quad (7)$$

$$\sum_{k \in M_i} a_{ik}^1 y_{ik}^1 + \sum_{k \in M_i} a_{ik}^2 y_{ik}^2 \leq t_i, \quad i \in J^2 \quad (8)$$

$$\left(1 - \sum_{j \in J^0 \setminus \{i\}} \sum_{k \in M_i} x_{ij}^k \right) \max_{k \in M_i} (b_{ik}) + \sum_{k \in M_i} b_{ik} \sum_{j \in J^0 \setminus \{i\}} x_{ij}^k \geq t_i, \quad i \in J^1 \quad (9)$$

$$\left(1 - \sum_{k \in M_i} (y_{ik}^1 + y_{ik}^2) \right) \max_{k \in M_i} (b_{ik}) + \sum_{k \in M_i} b_{ik}^1 y_{ik}^1 + \sum_{k \in M_i} b_{ik}^2 y_{ik}^2 \geq t_i, \quad i \in J^2 \quad (10)$$

$$x_{ij}^k \in \{0, 1\} \text{ for } i, j \in J^0, k \in M; y_{ik}^1, y_{ik}^2 \in \{0, 1\}$$

$$\text{for } i \in J^2, k \in M_i; \text{ and } \min_{k \in M_j} (a_{jk}) \leq t_j \leq \max_{k \in M_j} (b_{jk}) \text{ for } j \in J \quad (11)$$

The objective function (1) maximizes the priority- and benefit-weighted number of scheduled jobs. The weight vector π_p must be specified so that, if job j is scheduled, its contribution to the objective function exceeds the total contribution of all lower class jobs. Constraint (2) limits each job to be processed by at most one machine. It also ensures that, if a job is scheduled, it has no more than one successor, which might be the dummy job 0. Constraint (3) limits the number of initial jobs and, hence, the number of machines used, to at most m . In addition, it indirectly specifies that each machine can process at most one job at a time.

Constraint (4) ensures that the same machine processes every job in a scheduled sequence; i.e., if job j is assigned to machine k , both its predecessor and successor must be processed by machine k . This equality is usually referred to as the conservation of flow constraint. Inequality (5) is the compatibility requirements of consecutive jobs. It also ensures subtour elimination because it forces the service initiation times, t_i and t_j , of any two consecutive jobs i and j in a sequence on the same machine to be strictly increasing. Therefore, the sequence cannot include any of the previously assigned jobs. Constraint (6) ensures that, if job i is assigned to machine k , then it is processed in only one time window.

Constraints (7)–(10) enforce the time windows. The service initiation times must fall within two of these bounds depending on whether the job has one or two time windows. Note that t_j still has bounds of $\min_{k \in M_j} (a_{jk})$ and $\max_{k \in M_j} (b_{jk})$ even when it is not scheduled. The reason derives from the definition of μ_{ij} [see Constraint (5)], which is based on the largest and smallest values of the time windows of jobs i and j .

From a practical point of view, the formulation needs to be repeatedly solved in the following way to avoid dealing with excessively large values of π_p .

Step 1. Let $S_1 \equiv J_1$. Solve the problem for jobs $j \in S_1$ only and let the resulting objective function value be z^1 . Set $p = 2$.

Step 2. Let $S_p = S_{p-1} \cup J_p$. Solve the problem for jobs $j \in S_p$ only with added constraints that require the objective function value for all jobs $j \in S_h$ ($h = 1, \dots, p - 1$) to be at least as great as z^h , where z^h is the objective function value obtained when only jobs $j \in S_h$ were considered. These constraints can be written as

$$\sum_{i \in S_h} \pi_{P_i} \beta_i \left(\sum_{j \in J^0 \setminus \{i\}} \sum_{k \in M} x_{ij}^k \right) \geq z^h, \quad h = 1, \dots, p - 1 \quad (12)$$

Call the resulting objective function value z^p . Put $p \leftarrow p + 1$.

Step 3. Repeat Step 2 until $p > \rho$.

If machines can be partitioned into subsets such that each subset is homogeneous, the number of constraints can be reduced by redefining k as an index of a machine type. To see this, let M_k represent the set of machines of type $k \in K$ and recall that M is the set of all machines. As such, $M \equiv \bigcup_{k \in K} M_k$. The corresponding mixed-integer linear program is very similar to (1)–(11) except that M is replaced by K and constraint (3) is replaced by

$$\sum_{j \in J} x_{0j}^k \leq |M_k|, \quad k \in K \quad (13)$$

Model 2: Two Services for Each Job

In the TDRSS application, some requests may be for two services. These do not necessarily have to be on the same machine and, in fact, must be scheduled on different machines if they are required to be made simultaneously. To extend the model to account for this situation, let τ_i be the offset time between the start of service 1 and service 2 for job i ; τ can be negative, zero, or positive. For example, $\tau = -300$ indicates that service 2 must start exactly 300 time units before the start of service 1, $\tau = 0$ indicates that the two services must start simultaneously, while $\tau = 300$ indicates that service 2 starts exactly 300 time units after the start of service 1.

To handle this case, we split each two-service job into two jobs, one for service 1 and the other for service 2. In addition, we relabel the job indices such that if job i represents

service 1 of some two-service request, then job $i + 1$ must represent service 2 of that request. If i is service 1 of some two-service request, then the parameter β_i can be interpreted as the benefit received when both jobs i and $i + 1$ are scheduled. If only one of the jobs can be scheduled, the request is considered unsatisfied and no contribution is made to the objective function.

Let S_1 and S_2 be the set of jobs representing service 1 and 2, respectively. The augmented model is

$$\text{Maximize} \quad \sum_{i \in J_1} \pi_{P_i} \beta_i \left(\sum_{j \in J^0 \setminus \{i\}} \sum_{k \in M} x_{ij}^k \right) \quad (14)$$

subject to constraints (2)–(11),

$$\sum_{j \in J^0 \setminus \{i-1\}} \sum_{k \in M} x_{(i-1)j}^k - \sum_{j \in J^0 \setminus \{i\}} \sum_{k \in M} x_{ij}^k = 0, \quad i \in S_2 \quad (15)$$

$$t_{i-1} + \tau_{i-1} = t_i, \quad i \in S_2 \quad (16)$$

Constraint (15) ensures that if a job requiring two services is selected, then both are scheduled. Constraint (16) ensures the correct offset time between the start of service 1 and service 2.

5. BRANCH AND PRICE

Branch and price (B&P) has been implemented successfully for wide variety of vehicle routing/scheduling problems (Desrochers, Desrosiers, and Solomon [8], Desrosiers et al. [10]) and machine scheduling problems (Chan, Muriel, and Simchi-Levi [7]), Powell and Chen [19]) to name a few. In simplified terms, the general procedure can be summarized as follows (Barnhart et al. [4]).

The problem is first formulated as an integer program (IP) in which each restriction is represented by a set of algebraic constraints. Dantzig–Wolfe decomposition is then applied, resulting in a (restricted) set covering master problem that is solved with branch and bound. Bounds are computed at each node of the search tree by solving the linear relaxation of the set covering IPs; new columns are generated by solving integral pricing subproblems.

The benefit of the set covering reformulation is that, when the pricing subproblems are not totally unimodular, the LP bounds are usually much tighter than those obtained by solving the LP relaxation of the original constraint-based formulation. The effectiveness of the approach, though, depends on the number and difficulty of the subproblems since they will have to be solved many times.

In the remainder of this section, we describe how the constraint-based model defined by (1) – (11) is transformed into a master problem and a series of subproblems—one for each machine. We then discuss how each subproblem is

solved with our implementation of the generalized threshold algorithm. This is followed by the details of the branch-and-price algorithm whose major components include an initialization scheme for the master problem, two branching rules for constructing the search tree, and a lower-bounding procedure. Several examples are also given to illustrate the computations.

5.1. Set Packing Formulation

After applying Dantzig–Wolfe decomposition to PMSPTW, a set packing rather than a set covering problem results. To see how the decomposition is carried out, define a *partial schedule* as a feasible schedule on a single machine formed by a subset of the available jobs in N . Accordingly, a feasible solution to PMSPTW on m machines is simply m partial schedules with no intersecting jobs.

From the original formulation of PMSPTW with one service only for each job defined by (1)–(11), the set of feasible partial schedules on machine k is the set of points that satisfies (4)–(11) for k fixed. By applying Dantzig–Wolfe to the original formulation, the problem decomposes into a master problem consisting of (1)–(3) and m subproblems with feasible regions defined by (4)–(11).

Let Ω^k denote the set of all feasible partial schedules for machine k . For each job $j \in J$ and $s \in \Omega^k$, let $\delta_{js}^k = 1$ if schedule $s \in \Omega^k$ processes job j , and 0 otherwise. Let w_s^k be the total weight of the jobs processed on machine k using schedule s and define a binary variable $q_s^k = 1$ if schedule $s \in \Omega^k$ is selected, and 0 otherwise. With this notation, the master problem \mathcal{MP} is

$$\text{Maximize} \quad \sum_{k \in M} \sum_{s \in \Omega^k} w_s^k q_s^k \quad (17)$$

subject to

$$\sum_{k \in M} \sum_{s \in \Omega^k} \delta_{js}^k q_s^k \leq 1, \quad j \in J \quad (18)$$

$$\sum_{s \in \Omega^k} q_s^k \leq 1, \quad k \in M \quad (19)$$

$$q_s^k \in \{0, 1\}, \quad s \in \Omega^k, k \in M \quad (20)$$

If θ_j and ϕ_k are the dual variables associated with Eqs. (18) and (19), respectively, then the reduced cost of a variable q_s^k in \mathcal{MP} is

$$\bar{w}_s^k = w_s^k - \sum_{j \in J} \theta_j \delta_{js}^k - \phi_k \quad (21)$$

and the optimality conditions are

$$w_s^k - \sum_{j \in J} \theta_j \delta_{js}^k - \phi_k \leq 0, \forall k \in M, \quad s \in \Omega^k \quad (22)$$

Note that, as previously defined in Section 4, the priority-and benefit-weighted objective function coefficient of job j is $\pi_{P_j} \beta_j$. Therefore, the total weight of the jobs processed on machine k by schedule s is

$$w_s^k = \sum_{j \in J} \pi_{P_j} \beta_j \delta_{js}^k \quad (23)$$

Using Eq. (23) and multiplying through by (-1) , the optimality condition becomes

$$\sum_{j \in J} (\theta_j - \pi_{P_j} \beta_j) \delta_{js}^k + \phi_k \geq 0 \quad (24)$$

Because ϕ_k is common to all partial feasible schedules on k , it can be treated as a constant when solving the subproblems. Therefore, the subproblem associated with machine k is to find a feasible partial schedule $s \in \Omega^k$ with minimum total weight of jobs, where the weight of job j is $(\theta_j - \pi_{P_j} \beta_j)$. Optimality of the master problem is reached when the minimum total weight partial schedule of each machine $k \in M$ satisfies Eq. (24).

To convert the variables in the master problem back to the original variables, the following relationship can be used.

$$x_{ij}^k = \sum_{s \in \Omega^k} e_{ij}^s q_s^k \quad (25)$$

where $e_{ij}^s = 1$ if both i and j are included in schedule s and j is the immediate successor of i ; otherwise, $e_{ij}^s = 0$.

Because the original formulation and the set packing reformulation are both valid for PMSPTW, an optimal solution to one is optimal to the other. Nevertheless, when the relaxed version of the set packing model is solved, it is likely to produce a better bound, call it z_{CG} , than the LP relaxation, z_{LP} . This follows from the fact that, to get z_{CG} , we are optimizing over the convex hull of feasible points of (4)–(11) rather than over the LP relaxation of these constraints (see Wolsey [22]). Working with the set packing model is also more likely to yield feasible solutions because only convex combinations of feasible sequences are permitted.

During branch and bound, the definition of a *feasible* partial schedule needs to be extended to account for both explicit and implicit job-ordering restrictions. The former are those imposed by the precedence constraints that are part of the original problem, while the latter are those imposed by the branching constraints.

5.2. The Subproblem: Shortest Path Problem with Time Windows and 2-Cycle Elimination Procedure

The subproblem associated with machine k is to find a feasible partial schedule whose total job weights are

minimized; i.e.,

$$\text{Minimize} \quad \sum_{j \in I} (\theta_j - \pi_{P_j} \beta_j) x_{ij}^k + \phi_k \quad (26)$$

$$\text{subject to} \quad (4)\text{--}(11) \quad (27)$$

This is equivalent to the elementary shortest path problem with time windows (ESPPTW) which is NP-hard in the strong sense (Dror [11]) so the existence of a pseudo-polynomial algorithm is unlikely. Nevertheless, the relaxed version, the shortest path problem with time windows (SPPTW), where each node may be visited more than once, is NP-hard in the ordinary sense. Although cycles may exist in a solution to this problem, the time window constraints guarantee that all feasible paths are finite.

Powell and Chen [19] developed an $O(D^3)$ label correcting procedure for SPPTW that they called the generalized threshold algorithm (GTA). Empirically, it was shown to run faster than the $O(D^2)$ label setting algorithm of Desrochers and Soumis [9], where $D = \sum_{i \in V} (b_i - a_i + 1)$. Because of its better average performance, we chose GTA to solve the subproblems at each node of our B&P algorithm.

According to Desrochers, Desrosiers and Solomon [8], the quality of the LP bound of the set covering formulation of the master problem may be much improved by eliminating subproblem solutions that contain a 2-cycle. A path contains a 2-cycle if it revisits a node after visiting exactly one other node; i.e., at some point the path visits node x and then visits some other node y , and then immediately returns to node x . To improve the quality of the subproblem solutions, we have extended GTA to include 2-cycle elimination logic. The algorithm is described below.

Let $G = (V, A)$ be a directed graph, where $V = N \cup \{p, q\}$ is the set of nodes such that p is the source node and q is the sink node, and A is the set of arcs. Each arc $(i, j) \in A$ has a positive duration t_{ij} , which is the time to travel from i to j , and a cost c_{ij} . Each node $i \in V$ has a time window $[a_i, b_i]$ within which a visit can start. The objective is to find the least cost path between p and q while respecting the time windows of each node visited.

In the algorithm, a label (T_i^α, C_i^α) is assigned to each node, where T_i^α is the starting time of service at node i for path α and C_i^α is the cost of the path. When comparing two labels, there are two types of ordering relationship: Dominance ordering and lexicographic ordering.

DEFINITION 1: (T_1, C_1) *dominates* (T_2, C_2) , written as $(T_1, C_1) < (T_2, C_2)$, if and only if $T_1 \leq T_2$, $C_1 \leq C_2$ and at least one of the inequalities is strict.

DEFINITION 2: (T_1, C_1) is *lexicographically* less than (T_2, C_2) , written as $(T_1, C_1) \stackrel{L}{<} (T_2, C_2)$, if and only if $T_1 < T_2$; or $T_1 = T_2$ and $C_1 < C_2$.

DEFINITION 3: (T_1, C_1) is *lexicographically* less than or equal to (T_2, C_2) , denoted by $(T_1, C_1) \stackrel{L}{\leq} (T_2, C_2)$, if and only if $(T_1, C_1) \stackrel{L}{<} (T_2, C_2)$ or $(T_1, C_1) = (T_2, C_2)$. A label $(\hat{T}, \hat{C}) \in Q$ is the *lexicographic minimum label* of Q if $(\hat{T}, \hat{C}) \stackrel{L}{\leq} (T, C)$; $\forall (T, C) \in Q \setminus \{(\hat{T}, \hat{C})\}$.

DEFINITION 4: Let Q be a set of labels. A label $(T, C) \in Q$ is *efficient* with respect to Q if no other label in Q dominates it. A label for a node is said to be *efficient* if it is efficient with respect to the set of labels at the node. The path associated with the efficient label is the *efficient path* for the node.

Define $R_i = \cup_{\alpha} \{(T_i^\alpha, C_i^\alpha)\}$ be the set of efficient labels at node i . Given an efficient label (T_i, C_i) , an efficient path is the shortest path arriving at node i at time no later than T_i .

Let (T_{thres}, C_{thres}) be the threshold label. At each iteration of the algorithm, the following updating procedure is applied.

$$(T_{thres}^0, C_{thres}^0) = (T_{basic}, C_{basic})$$

$$(T_{thres}^{n+1}, C_{thres}^{n+1}) = (T_{thres}^n, C_{thres}^n) + (1, 1) + (T_{basic}, C_{basic})$$

where

$$(T_{basic}, C_{basic}) = (T_{avg}, C_{avg}) \times \alpha_1 / \alpha_2$$

such that

$$\alpha_2 = \min \left(50, \frac{|A|}{|V|} \right)$$

$$T_{avg} = \frac{1}{|A|} \sum_{(i,j) \in A} t_{ij}$$

$$C_{avg} = \frac{1}{|A|} \sum_{(i,j) \in A} c_{ij}$$

$$5 \leq \alpha_1 \leq 10$$

Note that (T_{basic}, C_{basic}) is computed from the characteristics of the network with an adjustment factor α_1 . Based on empirical results, choosing a value of α_1 between 5 and 10 is recommended.

5.3. GTA Algorithm

Let Q_1, Q_2 , and Q_3 be the three queues used by the algorithm for maintaining a candidate list L of labels. A queue is a first-in-first-out data structure where new objects are placed at one end and removed from the other. The general steps of the algorithm follow.

Step 0.

0.1. Construct $\Gamma(i) = \{j : (i, j) \in A\} \forall i \in V$; i.e., the set of nodes that immediately follow node i in the graph G .

0.2. Initialize the set of efficient labels for each node $i \in V$:

$$R_p = \{(T_p^1 = a_p, C_p^1 = 0)\}$$

$$R_i = \{(T_i^1 = a_i, C_i^1 = \infty)\}, \forall i \in V \setminus \{p\}$$

0.3. Initialize the three queues:

$$Q_1 = \{(T_p^1 = a_p, C_p^1 = 0)\}$$

$$Q_2 = Q_3 = \emptyset$$

0.4. Initialize the threshold label:

$$(T_{thres}, C_{thres}) = (T_{basic}, C_{basic})$$

Step 1. Select (T_i^α, C_i^α) from the bottom of Q_1 .

Step 2. (*Treat the label*) For each $j \in \Gamma(i)$.

2.1. If $T_i^\alpha + t_{ij} \leq b_j$ and selecting j does not create a 2-cycle in the efficient path associated with (T_i^α, C_i^α) ,

$$(T_j^{new}, C_j^{new}) = (\max\{a_j, T_i^\alpha + t_{ij}\}, C_i^\alpha + c_{ij})$$

2.2. If (T_j^{new}, C_j^{new}) is not dominated by any label in R_j ,

- For all labels $(T_j, C_j) \in R_j$, if $(T_j^{new}, C_j^{new}) < (T_j, C_j)$, remove (T_j, C_j) from R_j, Q_1, Q_2 , and Q_3
- $R_j = R_j \cup \{(T_j^{new}, C_j^{new})\}$
- If $(T_j^{new}, C_j^{new}) \stackrel{L}{\leq} (T_{thres}, C_{thres})$, add (T_j^{new}, C_j^{new}) to the bottom of Q_2 . Otherwise, add (T_j^{new}, C_j^{new}) to the bottom of Q_3 .

Step 3. Unless $Q_1 = \emptyset$, go to Step 1.

Step 4. (*Partition labels*) If $Q_2 \neq \emptyset$, transfer all labels in Q_2 to $Q_1, Q_2 = \emptyset$. Otherwise,

- if $Q_3 \neq \emptyset$, update the threshold label and get $(T_{thres}^{new}, C_{thres}^{new})$. If there is no label $(T, C) \in Q_3$: $(T, C) \stackrel{L}{\leq} (T_{thres}^{new}, C_{thres}^{new})$, find $(T_{lexmin}, C_{lexmin}) \in Q_3 : (T_{lexmin}, C_{lexmin}) \stackrel{L}{\leq} (T, C), \forall (T, C) \in Q_3 \setminus \{(T_{lexmin}, C_{lexmin})\}$, then update $(T_{thres}^{new}, C_{thres}^{new}) = (T_{lexmin}, C_{lexmin}) + (T_{basic}, C_{basic})$.
- Remove all labels $(T, C) \in Q_3 : (T, C) \stackrel{L}{\leq} (T_{thres}^{new}, C_{thres}^{new})$ and put them in Q_1 .

Step 5. Unless all three queues are empty, go to Step 3.

THEOREM 1: The modified generalized threshold algorithm with the 2-cycle exclusion Step (2.1) yields an optimal solution to the SPPTW without 2-cycles.

PROOF: The algorithm terminates with the optimal solution as long as no labels generated are dominated. Therefore,

the proof will show that the algorithm will not choose a job for processing that will lead to a dominated schedule. Assume that the partial schedule $(a, b, c, \dots, i, j, k, j, l, \dots)$ is feasible, where (j, k, j) is the first 2-cycle in the schedule. Let $(a, b, c, \dots, i, j, k, l)$ be path P_1 and $(a, b, c, \dots, i, k, j, l)$ be path P_2 . Assume that the algorithm generated P_1 but $(T_l^{P_2}, C_l^{P_2}) < (T_l^{P_1}, C_l^{P_1})$. Because the algorithm generated P_1 , $(T_j^{P_1}, C_j^{P_1}) < (T_k^{P_2}, C_k^{P_2})$. Also because the set of jobs preceding j in P_1 and k in P_2 are the same, $C_j^{P_1} = C_k^{P_2}$. From Definition 4, it follows that $T_j^{P_1} < T_k^{P_2}$, which implies that the finish time of k in P_1 is less than the finish time of j in P_2 ; therefore, $T_l^{P_1} \leq T_l^{P_2}$. Now, because the set of jobs preceding l is the same in both P_1 and P_2 , $C_l^{P_1} = C_l^{P_2}$. But from the assumption that $(T_l^{P_2}, C_l^{P_2}) < (T_l^{P_1}, C_l^{P_1})$, we have $T_l^{P_1} > T_l^{P_2}$, which is a contradiction. \square

5.4. GTA Example

An instance with three jobs is used to illustrate the 2-cycle elimination procedure. The data for the example are contained in Table 1, where a_i is the earliest start time of i and b_i is the latest start time of i .

First, the basic label is calculated as $(7.85714, -2.85714)$. Let p and q be the dummy source node and sink node, respectively. A directed graph is created when an arc exists between jobs i and j if job i can be processed immediately before job j . The resulting adjacency list of the graph is $p : \{1, 2, 3\}, 1 : \{p\}, 2 : \{p\}, 3 : \{1, p\}$.

ITERATION 1: Select the label $(0, 0)$ of node p from Q_1 (Step 1). After treating the other nodes (Step 2), the resulting labels are as follows:

Node 1 has a new label $(7, 0)$
Node 2 has a new label $(6, 0)$
Node 3 has a new label $(5, 0)$

Each of these labels is efficient and therefore placed in Q_3 .

ITERATION 2: Select label $(5, 0)$ of node 3 from Q_1 (Step 1). Node 1 and p are treated (Step 2) and the new labels are $(7, -1)$ and $(10, -1)$, respectively. Both are efficient and therefore, placed in Q_3 (Step 2.2).

Table 1. Data for GTA example.

i	p_i	a_i	b_i
1	2	7	8
2	3	6	7
3	3	5	7

- ITERATION 3: Select label (6, 0) of node 1 from Q_1 (Step 1). The only efficient label generated from node 1 is (10, -1) for node p (Step 2). Place it in Q_3 (Step 2.2).
- ITERATION 4: Select label (7, 0) of node 1 from Q_1 (Step 1). One efficient label is created which is (10, -1) for node p (Step 2).
- ITERATION 5: Q_1 is empty but Q_3 is not. Transfer all the labels from Q_3 to Q_1 (Step 4). The labels are (10, -1) at p from node 1, (10, -1) at p from node 2, (10, -1) at p from node 3, and (8, -1) at node 1 from node 3. Note that no labels were put in Q_2 so it has always been empty.
- ITERATION 6: Treating all four labels at node p yields no new label (Step 2). Next, label (8, -1) at node 1 is selected and treated (Step 2). A new label (10, -2) is created at p and placed in Q_3 (Step 2.2).
- ITERATION 7: Label (10, -2) at p is selected and treated (Step 2). No new label is created.
- ITERATION 8: All queues are empty (Step 5). The final path is $(p, 3, 1, q)$ with the length of 2.

5.5. Branch-and-Price Implementation

The B&P algorithm starts by solving the restricted master problem; i.e., the linear programming relaxation of a restricted version of the original set packing model Eqs. (17)–(20). In the development and exploration of the search tree, if the solution at some node is not integral or contains a cycle, an active problem P_i is chosen using the search function s . If the current upper bound $u(P_i)$ is less than or equal to the objective value of the incumbent solution, z^{best} , the node is fathomed and the next active problem is selected. Otherwise, the problem is set up with the branching restriction of the current node P_i .

If no improvement in the incumbent is realized in Ψ consecutive iterations, our greedy randomized adaptive search procedure (GRASP) is run in hopes of finding a new lower bound (Rojanasoonthon et al. [21]). The incumbent is updated if a better solution is found. Next, the upper bound is recalculated by solving the master problem using column generation. If the original variables x_{ij}^k in the solution are all integral, we check to see whether there are any cycles in the schedules. If no cycles are found and the objective function value is better than the incumbent z^{best} , then the incumbent is updated. In contrast, if the solution is not integral or some cycles exist and the upper bound shows that this node can still lead to a better solution than the incumbent, then the node is partitioned and the set of active nodes \mathcal{A} is updated. The two nodes created by partitioning inherit the upper bound of their parent.

The following notation is used to describe the algorithm.

- \mathcal{N} = set of partial problems still open; i.e., set of all nodes currently in the search tree.
- \mathcal{A} = set of active nodes, $\mathcal{A} \subset \mathcal{N}$.
- \mathcal{P} = set of all possible partial problems, where $P_i \in \mathcal{P}$ is the i^{th} partial problem in \mathcal{P} .
- z^{best} = objective function value of the incumbent solution.
- \mathcal{I} = incumbent solution.
- s = search function; i.e., the procedure for selecting an active node for immediate exploration.
- u = upper bound.
- $G(P_i)$ = objective function value of the GRASP solution for P_i .
- ψ = number of iterations without any improvement in z^{best} .
- Ψ = frequency of GRASP calls.

The algorithm is summarized in Figure 1. In the following subsections, we further describe the branching schemes, how to apply the branching restrictions to the subproblems, solving the master problem, what information to store at each node, how a node is selected from the pool of active nodes,

PROCEDURE BRANCH AND PRICE

```

Initialize:  $\mathcal{A} = P_0, \mathcal{N} = P_0, \mathcal{I} = \emptyset, \psi = 0$ . Run GRASP to get initial feasible solution; set  $z^{best} = G(P_0)$ .
while ( $\mathcal{A} \neq \emptyset$ ) do begin
   $P_i = s(\mathcal{A}); \mathcal{A} = \mathcal{A} \setminus P_i$ .
  if ( $u(P_i) \leq z^{best}$ ) (UB-test)
    continue
  Set up problem with branching restrictions.
  if ( $\psi \geq \Psi$ )
    Run GRASP.
    if ( $G(P_i) > z^{best}$ )
      Update  $z^{best}$  and  $\mathcal{I}$ .
       $\psi = 0$ .
  Calculate  $u(P_i)$  using column generation.
  if  $x_{ij}^k$ 's are integers
    if  $u(P_i) > z^{best}$  and no cycles exist in active columns,
      Update  $z^{best}$  and  $\mathcal{I}$ .
  if ( $u(P_i) > z^{best}$ ) (UB-test)
    if cycle exist in active columns,
      Time window branching.
    else if  $x_{ij}^k$ 's are not integer
      SOS branching.
    Add new nodes to  $\mathcal{A}$ .
  else
    Terminate  $P_i$ .
   $\psi = \psi + 1$ .
end while

```

Figure 1. Overview of branch-and-price logic for PMSPTW.

the data structure used, and, finally, the lower-bounding procedure.

5.6. Branching Schemes

Both time window branching and SOS branching are used by the B&P algorithm. Each serves a unique purpose and has proven to be an effective complement of the other.

5.6.1. Time Window Branching

In the current implementation, time window branching is applied when an active column in the master problem at node κ in the search tree (\mathcal{MP}^κ) contains a cycle. An active column is one whose associated q_s^k variable is nonzero; i.e., the column is included in the current solution of \mathcal{MP}^κ . Given a solution to the LP relaxation, \mathcal{LP}^κ , the algorithm searches for a job (1) with the highest priority weight, (2) that is processed the most times, and (3) that has the highest number of time windows (maximum of 2), in this order. Given such a job, its time window is partitioned using the following rules.

- If the job has two discrete time windows, create two descendent nodes, one for each time window.
- If the job j has only one time window on machine k , (a_{jk}, b_{jk}) , check for the following two cases. Let $\sigma_j = b_{jk} - a_{jk}$ be the slack in the time window of job j . Case I: the current time window has slack $\sigma_j = 1$. Now create the left child with the time window $[a_{jk}, a_{jk}]$ and the right child with the time window $[b_{jk}, b_{jk}]$. Case II: the time window has slack greater than 1. Create the left child with the time window $[a_{jk}, a_{jk} + \lfloor \frac{\sigma_j}{2} \rfloor]$ and the right child with the time window $[a_{jk} + \lfloor \frac{\sigma_j}{2} \rfloor, b_{jk}]$.

Time window branching for job j on a machine is exhausted once the slack $\sigma_j = 0$. The method is summarized in Figure 2.

5.6.2. Special Ordered Set Branching

Special ordered set (SOS) branching is based on the current values of the components of Eq. (2), $\sum_{j \in J} x_{ij}^k \leq 1$, which limits the number of predecessors of i to at most 1. If the solution to the LP relaxation is fractional, the algorithm searches for $\sum_j x_{ij}^k$ that is most fractional; i.e., the algorithm searches for the i and k indices whose $\sum_j x_{ij}^k$ is closest to 0.5. Given such an i and k , two branches are created. Let \bar{J} be a subset of J in which all variables x_{ij}^k are free, $j' \in \bar{J}$; i.e., none are fixed to zero due to previous restrictions on the path from the root node to the current node. Let \bar{J}^1 and \bar{J}^2 be a

PROCEDURE TIME WINDOW BRANCHING

begin

Find jobs visited multiple times in the solution.

Select a job j on machine k according to the following respective criterion: largest weight, number of visits, and number of time windows.

if j has two active time windows, $[a_{jk}^1, b_{jk}^1]$ and $[a_{jk}^2, b_{jk}^2]$,
create left child with time window $[a_{jk}^1, b_{jk}^1]$ and right child with $[a_{jk}^2, b_{jk}^2]$.

if j has only one active time window, $[a_{jk}, b_{jk}]$,
create left child with time window $[a_{jk}, a_{jk} + \lfloor \frac{\sigma}{2} \rfloor]$ and right child with $[a_{jk} + \lfloor \frac{\sigma}{2} \rfloor, b_{jk}]$.

end

Figure 2. Overview of time window branching for PMSPTW.

partition of the set \bar{J} such that $\bar{J}^1 \cup \bar{J}^2 = \bar{J}$ and $\bar{J}^1 \cap \bar{J}^2 = \emptyset$. SOS branching results in a left descendent with the restriction $\sum_{j \in \bar{J}^1} x_{ij}^k = 0$, which implies $\sum_{j \in \bar{J}^2} x_{ij}^k \leq 1$, and a right descendent with the restriction $\sum_{j \in \bar{J}^2} x_{ij}^k = 0$, which implies $\sum_{j \in \bar{J}^1} x_{ij}^k \leq 1$. To keep these two nodes balanced, \bar{J} is partitioned so that $\sum_{j \in \bar{J}^1} x_{ij}^k \cong \sum_{j \in \bar{J}^2} x_{ij}^k$ and the number of free x_{ij}^k 's whose current value is zero is approximately the same in \bar{J}^1 and \bar{J}^2 .

For the special case in which the inequality $\sum_j x_{ij}^k \leq 1$ has only one component, the left node is created with $x_{ij}^k = 0$ and the right node is created with $x_{ij}^k = 1$ if either i or j is processed on machine k . SOS branching for job j on machine k is exhausted when the corresponding inequality becomes empty. A more detailed discussion on how to enforce the branching requirements while solving the subproblem is presented in Section 5.7. Figure 3 summarizes the procedure.

PROCEDURE SOS BRANCHING

begin

Find i and k with the most fractional $\sum_j x_{ij}^k$ in the solution.

if \exists only one free variable in $\sum_j x_{ij}^k$,
create left child with restriction $x_{ij}^k = 0$ and right child with $x_{ij}^k = 1$ if either i or j is processed on machine k .

if \exists more than one free variable in $\sum_j x_{ij}^k$,
create left child with $\sum_{j \in \bar{J}^1} x_{ij}^k = 0$ and right child with $\sum_{j \in \bar{J}^2} x_{ij}^k = 0$, where \bar{J}^1 and \bar{J}^2 are such that $\sum_{j \in \bar{J}^1} x_{ij}^k \cong \sum_{j \in \bar{J}^2} x_{ij}^k$ and both have approximately the same number of free variables at zero.

end

Figure 3. Overview of SOS branching for PMSPTW.

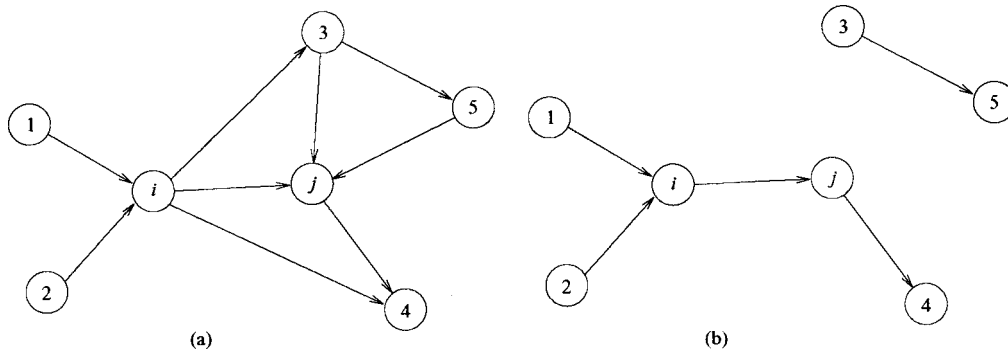


Figure 4. Example of subproblem network: (a) before applying $x_{ij}^k = 1$ requirement; (b) after applying $x_{ij}^k = 1$ requirement.

5.7. Application of the Branching Restrictions

This section discusses how the branching restrictions associated with a given master problem \mathcal{MP}^k are enforced when solving the SPPTW subproblems. We must consider the following two cases.

5.7.1. Time Window Branching

The time window branching restriction is applied in a straightforward manner by temporarily modifying the time window information of jobs. The algorithm collects the relevant branching restrictions of all predecessor nodes and updates the time windows of the associated jobs before solving SPPTW or running the GRASP.

5.7.2. Special Ordered Set Branching

When the SOS inequality has more than one free variable, the branching requirements are to set some x_{ij}^k to 0. Setting $x_{ij}^k = 0$ can be implemented directly in the subproblem by removing the corresponding arc in the network of machine k . As a result, machine k will not be able to process job i immediately before j . For the special case where the SOS inequality has only one free variable, the subproblem is modified in exactly the same manner as when the restriction is $x_{ij}^k = 0$.

The complementary restriction of setting $x_{ij}^k = 1$ is more difficult to handle. In this case, job i must be the immediate predecessor of job j on machine k , a restriction that cannot be enforced in an efficient way in the subproblem. For example, solving two SPPTWs, one from the source node p to node i and the second from node j to the sink node q does not necessarily provide the shortest path from p to q through (i, j) .

Instead, we modify the subproblem network by removing all arcs (i, l) for $l \neq j$ and (l, j) for $l \neq i$. This is equivalent to setting $x_{il}^k = 0$ for $l \neq j$ and $x_{lj}^k = 0$ for $l \neq i$. Hence, if either i or j is processed by k , both have to be processed

and i must immediately precede j . Although this implementation is less restrictive than setting $x_{ij}^k = 1$, it leads to a full partition of the feasible region. As the tree grows, all feasible sequences of interest will eventually be generated. At some point in the enumeration the remaining arcs in a subproblem will consist only of those in a particular sequence. Because no feasible sequence is excluded by this branching scheme, an optimal solution will eventually emerge. Its validity is formally stated below. Figure 4(a) shows the subproblem network before applying the restriction $x_{ij}^k = 1$ and Figure 4(b) shows the modified network after the restriction is applied.

PROPOSITION 1: The branching restriction $x_{ij}^k = 1$ imposed indirectly by removing all arcs (i, l) for $l \neq j$ and (l, j) for $l \neq i$ from the subproblem network, leads to a complete partition of the feasible region when coupled with its complement $x_{ij}^k = 0$. This scheme is equivalent to setting $x_{il}^k = 0$ for $l \neq j$ and $x_{lj}^k = 0$ for $l \neq i$.

Note that when running the GRASP to find feasible solutions at some node in the search tree, the SOS restrictions are enforced only during phase 1—construction. Because the idea is to perturb the GRASP in an attempt to find a feasible solution better than the incumbent, no checking is performed when running the local search in phase 2—improvement; i.e., local search moves are not required to respect the SOS branching restrictions.

5.8. Master Problem Initialization and Column Modification

At the root node of the search tree, a feasible solution is first obtained by running the GRASP. The corresponding schedules, one per machine, are used to initialize the LP associated with \mathcal{MP}^1 . At subsequent nodes, the initial columns of \mathcal{MP}^k are those inherited from its immediate predecessor; however, some of them may turn out to be infeasible due to the new

branching restrictions. Rather than removing the entire column to regain feasibility, a sequential search is conducted to find and remove those jobs that are the cause of the problem. Note that a column will never become infeasible with respect to the time window constraints when jobs are removed.

For time window branching, if the new restrictions at a node cause a column to be infeasible, then the job j associated with the violation is removed. This guarantees a feasible sequence since the column was originally feasible for the parent node and only became infeasible when the time window of j was changed. Removing j from the sequence allows the remaining jobs to shift back to their original starting times.

For SOS branching, the algorithm sequentially removes a job j that is an immediate successor to i on k if the restriction is $x_{ij}^k = 0$. The process is repeated until no violations exist. In the rare case in which all the jobs are removed from all columns, an initial column is generated with a single job whose weight is the largest among those that are not restricted from being the last job in the sequence.

Another parameter of the upper-bounding procedure is the number of columns added to the master problem each time a subproblem is solved. This number is set to 2 in the implementation so at most $2 \times |M|$ columns can be added at each node of the search tree.

5.9. Information Stored at Each Node

Storing detailed information at a node can make the B&P algorithm more efficient, but it can become a disadvantage in terms of memory usage as the size of the search tree grows. To achieve a balance between speed and memory, we store the following information at each node κ .

1. Parent and children node information $\Lambda^\kappa = \{\lambda_p^\kappa, \lambda_l^\kappa, \lambda_r^\kappa\}$, where λ_p^κ is the parent node, λ_l^κ is the left child, and λ_r^κ is the right child.
2. The job sequence σ_k^κ for each column associated with machine k in \mathcal{MP}^κ .
3. The complete LP associated with the node, \mathcal{LP}^κ . This is only stored temporarily and removed once node κ is partitioned.
4. Branching restrictions $\mathcal{B}^\kappa = \{\beta_{TW}^\kappa, \beta_{SOS}^\kappa\}$, where β_{TW}^κ is the time window branching restriction $[a_{ik}^{TW}, b_{ik}^{TW}]$ for job i on machine k and β_{SOS}^κ is the SOS branching restriction associated with $\sum_{j \in J} x_{ij}^k \leq 1$. To track the status of each variable in this inequality, we define a vector $v^\kappa = (v_1^\kappa, v_2^\kappa, \dots, v_{|J|}^\kappa)$, where v_j^κ takes on three values: $v_j^\kappa = 0$ if x_{ij}^k is free, $v_j^\kappa = 1$ if $x_{ij}^k = 0$, and $v_j^\kappa = 2$ if $x_{ij}^k = 1$.
5. The upper bound u^κ .
6. Node identification κ , where $\kappa = 1$ indicates the root node.

In summary form, the information stored at node κ is $(\Lambda^\kappa, \sigma^\kappa, \mathcal{LP}^\kappa, \mathcal{B}^\kappa, u^\kappa)$. Note that the branching information at a node only contains the restriction imposed by the partitioning decision at its parent node. To obtain the entire set of restrictions for a node, those of all its predecessors are collected. Let N^κ be the set of all the nodes on the path that joins κ and the root node. The restrictions at κ are $\{\mathcal{B}_i^\kappa : i \in N^\kappa\}$.

5.10. Node Selection

The best bound criterion is used to select the next node for processing from the active node pool \mathcal{A} . This method was chosen because it is computationally inexpensive and offers the prospect of finding good feasible solutions quickly. It also is more balanced and with fewer levels compared to, say, depth-first search.

5.11. Data Structures

To do the best bound search efficiently during the node selection step, another data structure is used to store the set of active nodes \mathcal{A} . These nodes are sorted according to their upper bound values and stored as an AVL tree (see Horowitz, Sahni, and Anderson-Freed [15]). The complexity of inserting, deleting, and finding a node in an AVL tree is $O(\log n)$. Retrieving the node associated with the maximum or minimum bound is also $O(\log n)$.

5.12. Lower-Bounding Procedure

The GRASP is called periodically in an attempt to convert a fractional solution at a node into a feasible schedule. The logic used for this purpose is based on the frequency with which the incumbent is updated during B&P. When no improvement in the incumbent is realized in Ψ consecutive iterations, the GRASP is called with the following parameters: an RCL size of 3, 1 iteration of phase 1, and 100 iterations of phase 2. Currently, $\Psi = 1$ due to the low computational burden of the procedure.

5.13. Branch-and-Price Example

A 20-job, 6-machine instance is used to demonstrate the B&P computations. Table 2 includes all the input data, where p_i is the processing time of job i , and a_{ik} and b_{ik} are the earliest and latest start times of job i on machine k , respectively. If the a_{ik} and b_{ik} are not defined for job i , it means that i cannot be processed by machine k .

Branch and price starts by solving the restricted master problem with an initial set of columns provided by the GRASP, one for each machine. For the example, the six columns in terms of job sequences are: machine 1: (5,13), machine 2: (7,4), machine 3:(8,15,16), machine 4:(6,2,9),

Table 2. Input data for 20-job, 6-machine example.

i	p_i	a_{i1}	b_{i1}	a_{i2}	b_{i2}	a_{i3}	b_{i3}	a_{i4}	b_{i4}	a_{i5}	b_{i5}	a_{i6}	b_{i6}
1	1017	1835	2038	2395	2524								
2	1012	361	1293	1640	1723	17	726	1450	1825	195	773	1672	1930
3	1800	290	545	395	1727	391	584	566	1717				
4	819	1208	2010	1825	1875								
5	1615	598	598	1153	1153	1356	1356	1347	1347				
6	467					497	535	719	720				
7	971	1692	1692	576	576								
8	64			911	913	2142	2162						
9	639	2238	2305	2338	2467	1609	1666	2657	2756				
10	1646									128	374		
11	486									494	940	1813	1963
12	350	956	1079	1964	2021	1517	1618	2809	3019	582	1512	1974	2877
13	772	2752	2752	1346	1346	1007	1007	2817	2817				
14	689									680	1241	597	733
15	660	989	1243	844	903	1276	1763	2812	2829				
16	1525	599	1202	67	363	514	2008	317	702				
17	434									2989	2997	1451	1466
18	542									2990	3015	1431	1720
19	638					596	940	2299	2358				
20	1921	1435	1435	1544	1544	1225	1225	1113	1113				

Table 3. Computational results for 20-job, 2-machine instances (B&P).

Name ^a	Optimal solution	LP % gap at root	Columns generated at root	Average columns per node	Node with first integer solution	Total nodes	GRASP improves	GRASP calls
lpltw1	16	0	34	34	-1	1	0	0
lpltw2	16	0	32	32	-1	1	0	0
lpltw3	16	0	34	34	-1	1	0	0
lpltw4	15	3.33	30	30	-1	1	0	0
lpltw5	15	0	27	27	-1	1	0	0
lpttw1	12	0	23	23	-1	1	0	0
lpttw2	10	0	13	13	-1	1	0	0
lpttw3	13	0	29	29	-1	1	0	0
lpttw4	12	0	26	26	-1	1	0	0
lpttw5	12	0	28	28	-1	1	0	0
rand1	13	0	28	28	-1	1	0	0
rand2	15	0	25	25	-1	1	0	0
rand3	14	0	22	22	-1	1	0	0
rand4	17	0	42	42	1	1	0	0
rand5	16	0	29	29	-1	1	0	0
spltw1	20	0	37	37	-1	1	0	0
spltw2	17	0	44	44	-1	1	0	0
spltw3	19	0.65	60	60	-1	1	0	0
spltw4	14	0	18	18	-1	1	0	0
spltw5	19	3.95	53	53	-1	1	0	0
spttw1	14	0	27	27	-1	1	0	0
spttw2	13	0	26	26	-1	1	0	0
spttw3	15	0	38	38	-1	1	0	0
spttw4	11	0	19	19	-1	1	0	0
spttw5	14	0	28	28	-1	1	0	0
Average	14.72	0.32	30.88	30.88	-0.92	1	0	0

^a lpltw, long processing time and loose time windows; lpttw, long processing time and tight time windows; rand, random processing time and time windows; spltw, short processing time and loose time windows; spttw, short processing time and tight time windows.

Table 4. Computational times for 20-job, 2-machine instances (s).

Name	Total time	Time to best solution	Time at root	Time for GRASP	Average time per node
lpltw1	0.05	0	0.05	0	0.05
lpltw2	0.05	0	0.05	0	0.05
lpltw3	0.06	0	0.06	0	0.06
lpltw4	0.06	0	0.06	0	0.06
lpltw5	0.05	0	0.05	0	0.05
lpttw1	0.02	0	0.02	0	0.02
lpttw2	0.01	0	0.01	0	0.01
lpttw3	0.03	0	0.02	0	0.02
lpttw4	0.03	0	0.03	0	0.03
lpttw5	0.03	0	0.02	0	0.02
rand1	0.03	0	0.03	0	0.03
rand2	0.04	0	0.04	0	0.04
rand3	0.03	0	0.03	0	0.03
rand4	0.07	0.07	0.07	0	0.07
rand5	0.06	0	0.05	0	0.05
spltw1	0.07	0	0.06	0	0.06
spltw2	0.09	0	0.09	0	0.09
spltw3	0.16	0	0.16	0	0.16
spltw4	0.03	0	0.02	0	0.02
spltw5	0.2	0	0.18	0	0.18
spttw1	0.03	0	0.03	0	0.03
spttw2	0.02	0	0.02	0	0.02
spttw3	0.04	0	0.04	0	0.04
spttw4	0.02	0	0.01	0	0.01
spttw5	0.02	0	0.02	0	0.02
Average	0.052	0.0028	0.0488	0	0.0488

machine 5: (20,12,18), and machine 6: (14,17,11). These results imply that $z^{best} = 16$.

ITERATION 1: At node 1 (root), a solution to the master problem \mathcal{MP}^1 was found after 40 more columns were added. The objective function value of the LP relaxation is $u^1 = 17$. Some of the variables in the solution are fractional but there are no cycles in any of the active columns so SOS branching is invoked. The inequality $\sum_{j \in J} x_{2,j}^1 \leq 1$ is selected and two descendent nodes are created based on the fact that $x_{2,4}^1 = 0.5$ is the only nonzero variable in the inequality and all the other variables are free. The left child, indexed as node 2, has the branching restriction $x_{2,j}^1 = 0$ for all $j \in \{1, 2, 3, 5, 6, 7, 8, 9, 10\}$ and the right child, indexed as node 3, has the restriction $x_{2,j}^1 = 0$ for all $j \in \{4, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\}$. Both nodes 2 and 3 are put in the active pool \mathcal{A} . Finally, the GRASP is run but did not provide an improved feasible solution.

ITERATION 2: Examining the active pool $\mathcal{A} = \{2, 3\}$, we see that $u^2 = 17$ and $u^3 = 17$ so there is no preference between nodes 2 and 3. The search function $s(\mathcal{A})$ arbitrarily selected node 3. \mathcal{MP}^3 is then constructed and solved to optimality after adding 36 more columns bringing the total to 50. The rounded objective function of the LP relaxation is $u^3 = 17$. Although the solution is fractional, there are no cycles in any of the active columns so SOS branching is invoked. The inequality $\sum_{j \in J} x_{12,j}^5 \leq 1$ is selected and two descendent nodes are created based on the fact that $x_{12,18}^6 = 0.5$ is the only relevant nonzero variable. The left child, indexed as node 4, has the branching restriction $x_{12,j}^6 = 0$ for all $j \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and the right child, indexed as node 5, has the restriction $x_{12,j}^6 = 0$ for all $j \in \{10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\}$. Both node 4 and 5 are put in \mathcal{A} and the GRASP is run producing a solution of 16, which is equal to z^{best} .

Table 5. Computational results for 30-job, 6-machine instances (B&P).

Name	Optimal solution	LP % gap at root	Columns generated at root	Average columns per node	Node with first integer solution	Total nodes	GRASP improves	GRASP calls
lpltw1	28	2.21	144	208.43	1901	1901	1	317
lpltw2	25	2.40	129	147.00	8	11	1	2
lpltw3	23	1.09	88	118.67	29	29	0	6
lpltw4	27*	4.87*	125	^a	^a	12250	^a	^a
lpltw5	25*	9.2*	104	^a	^a	31243	^a	^a
lpttw1	23	2.61	94	94.00	-1	1	0	0
lpttw2	21	2.38	73	73.00	5	5	1	1
lpttw3	23	4.01	78	78.00	-1	1	0	0
lpttw4	26	0.27	93	109.37	17	71	1	14
lpttw5	21	0.00	53	53.00	1	1	0	0
rand1	26	1.91	125	132.50	11	11	1	2
rand2	27	2.47	123	129.00	5	5	1	1
rand3	26	0.00	108	108.00	-1	1	0	0
rand4	27	3.22	131	134.33	11	11	1	2
rand5	29	1.08	131	177.70	2	4605	1	767
spltw1	29	0.00	112	135.00	17	17	0	3
spltw2	29	2.02	158	224.87	119	119	1	20
spltw3	29	2.50	104	87.33	5	3	1	1
spltw4	29*	3.45*	107	^a	^a	31077	^a	^a
spltw5	30	0.00	104	104.00	-1	1	0	0
spttw1	23	3.56	80	80.00	-1	1	0	0
spttw2	26	1.28	106	106.00	-1	1	0	0
spttw3	29	0.00	87	126.83	21	21	1	4
spttw4	28	0.00	78	93.00	11	11	0	2
spttw5	26	2.23	91	93.67	5	5	1	1
Average ^b	26.1	1.6	104.1	118.8	98.3	310.6	0.6	52

^a Branch and price ran out of memory before producing an optimal solution. Values in column 2 and 3 are based on the initial GRASP solution.

^b Average values based only on instances that were solved.

ITERATION 3: Node 5 is selected next by the search function, $s(\mathcal{A}) = \text{node } 5$. \mathcal{MP}^5 is constructed and solved to optimality with 54 columns. The objective function value is 17, all the variables are integral, and no cycle exists in the solution. Therefore, the incumbent is updated by setting $z^{best} = 17$ and the node is fathomed. The corresponding schedule is (2, 17, 13) on machine 1, (3, 1) on machine 2, (19, 15, 16) on machine 3, (6, 8, 9) on machine 4, (20, 18) on machine 5, and (14, 17, 11, 12) on machine 6.

ITERATION 4: The remaining nodes in \mathcal{A} are 2 and 4. Both have an upper bound of 17 so they are fathomed and the algorithm terminates.

The full problem was solved in 0.27 s. As we shall see in the next section, this type of performance is typical of problem instances with up to 50 jobs and 2 machines.

6. COMPUTATIONAL RESULTS

The B&P algorithm was implemented in C++ using the gcc compiler version 3.3 on a Dell Precision 530 workstation running SuSE linux 8.2. The Dell has dual 1.8 GHz Xeon processors with 1 GB of RAM but only one processor was used in the computations. CPU times were collected using the C function *clock()* and CPLEX version 7.5 was used to solve the linear program master problems.

6.1. Data Sets

For testing purposes, five data sets containing five instances each were randomly generated for different problem sizes using the same scheme described in Rojanasoonthon, Bard, and Reddy [21]. The data sets fell into the following categories.

1. Short processing time and loose time windows (spltw)
2. Short processing time and tight time windows (spttw)

Table 6. Computational times for 30-job, 6-machine instances (s).

Name	Total time	Time to best solution	Time at root	Time for GRASP	Average time per node
lpltw1	237.95	237.81	0.5	190.08	0.05
lpltw2	2.18	2.14	0.32	1.44	0.12
lpltw3	7.19	7.19	0.36	5.67	0.08
lpltw4	5844	a	0.42	a	a
lpltw5	5760	a	0.32	a	a
lpttw1	0.07	0	0.07	0	0.07
lpttw2	0.65	0.64	0.05	0.58	0.02
lpttw3	0.05	0	0.05	0	0.05
lpttw4	9.51	9.51	0.08	8.76	0.01
lpttw5	0.04	0.04	0.04	0	0.04
rand1	1.96	1.92	0.23	1.43	0.08
rand2	0.89	0.88	0.15	0.7	0.06
rand3	0.07	0	0.07	0	0.07
rand4	1.88	1.84	0.18	1.53	0.06
rand5	761.42	761.35	0.41	664.16	0.04
spltw1	3.64	3.64	0.35	2.57	0.11
spltw2	30.9	30.63	1.26	18.04	0.21
spltw3	3.64	3.64	1.2	2.6	0.12
spltw4	5822	a	0.5	a	a
spltw5	0.16	0	0.16	0	0.16
spttw1	0.05	0	0.05	0	0.05
spttw2	0.07	0	0.07	0	0.07
spttw3	2.87	2.84	0.06	2.59	0.02
spttw4	1.55	1.55	0.06	1.4	0.02
spttw5	0.63	0.6	0.09	0.5	0.04
Average ^b	48.52	48.46	0.28	41.00	0.07

^a Branch and price ran out of memory before producing an optimal solution.

^b Average values based only on instances that were solved.

3. Long processing time and loose time windows (lpltw)
4. Long processing time and tight time windows (lpttw)
5. Random instances (rand)

Although categories 2 and 4 best reflect the real data, our intent was to establish the effectiveness of the algorithm over a wide range of parameter values. We were also interested in determining how large a problem the algorithm could solve. The problem sizes investigated were: 20 jobs and 2 machines, 20 jobs and 6 machines, 30 jobs and 2 machines, 30 jobs and 6 machines, 50 jobs and 2 machines, 50 jobs and 3 machines, and 100 jobs and 2 machines. In all instances, the objective function coefficient β_i was fixed at 1 for all $i \in J$, implying that each job was equally important within its priority class. Depending on the scenario, other settings might be more appropriate. When the benefit of a job is proportional to its length, for example, β_i should be defined accordingly.

In the generation process, each job had a 10% chance of having two time windows, which were either loose or tight depending on the data set. The planning period was set at 3600 s and the processing times and time windows

were scaled to reflect the number of machines in a particular instance. In general terms, we first determined the total time available for processing to be $3600 \times$ number of machines, and then divided the result by the number of jobs. This gave us the time per job. We then created a range that covered the time per job and randomly selected a processing time based on a probability distribution derived from the original data. As an example, if there were 20 jobs and 2 machines, then the total time available = $3600 \times 2 = 7200$ s and the time per job = 360 s. The range used in this instance was 10 to 720 s. A similar procedure was used to generate the time windows.

In all data sets, the machines were assumed to be heterogeneous so the processing time and time window(s) of a job were a function of the machine. The computations were halted when a zero optimality gap was reached or when an “out of memory” error message was reported.

6.2. Output

The results are summarized in Tables 3 to 8. Algorithmic performance for the 20-job, 2-machine instances is reported in Table 3. The column headings are (1) problem name, which

Table 7. Computational results for 100-job, 2-machine instances (B&P).

Name	Optimal solution	LP % gap at root	Columns generated at root	Average columns per node	Node with first integer solution	Total nodes	GRASP improves	GRASP calls
lpltw1	86*	3.69*	2201	a	a	1120	a	a
lpltw2	84	0.60	2147	2374.44	17	17	0	3
lpltw3	80*	2.47*	2144	a	a	1263	a	a
lpltw4	84	0.60	2149	2597.17	5	361	1	69
lpltw5	83	0.80	2015	2335.88	79	79	1	14
lpttw1	60	0.00	79	86.5	3	3	0	0
lpttw2	66	0.00	118	451	5	5	1	1
lpttw3	61	0.00	59	75.6667	5	5	0	1
lpttw4	64	0.00	79	290.5	11	11	1	2
lpttw5	57	0.00	28	28	1	1	0	0
rand1	76	0.00	122	122	1	1	0	0
rand2	73	0.68	2095	2095	-1	1	0	0
rand3	77	0.00	170	945.5	7	7	0	1
rand4	71	0.00	144	1598.4	5	9	1	1
rand5	75	0.00	1974	1974	1	1	0	0
spltw1	84	0.89	2115	2306.17	23	23	0	4
spltw2	92	0.72	2312	2806.93	11	59	1	10
spltw3	91	1.65	2118	2424.29	53	83	0	14
spltw4	87	0.92	2196	3045.78	221	221	1	37
spltw5	88	0.88	2319	2482.5	15	15	0	2
spttw1	69	0.00	83	83	-1	1	0	0
spttw2	70	0.00	88	88	-1	1	0	0
spttw3	65	0.00	98	98	1	1	0	0
spttw4	66	0.00	71	109.4	9	9	0	1
spttw5	66	0.00	35	35	1	1	0	0
Average ^b	73.9	0.32	930.32	1185.39	20.64	40.82	0.32	7.14

^a Branch and price ran out of memory before producing an optimal solution. Values in column 2 and 3 are based on the initial GRASP solution.

^b Average values based only on instances that were solved.

reveals some of its characteristics, (2) optimal objective function value, (3) percentage gap between the LP objective value at the root node and the optimal objective value, (4) number of columns generated at the root node to solve the LP relaxation, (5) average number of columns generated per node, (6) index of the node at which the first integer solution was found either by GRASP or by the B&P algorithm, (7) total number of nodes in the search tree, (8) number of times GRASP produced a new incumbent solution, and (9) total number of times GRASP was called after initialization. With respect to heading (4), an entry of -1 indicates that the initial GRASP solution was optimal and that B&P was able to confirm this at the root node after solving the LP relaxation.

Table 4 contains the computation times (seconds) for the various components of the algorithm for the 20-job, 2-machine instances. The column headings are (1) problem name, (2) total CPU time, (3) time to find best feasible (optimal) solution, (4) time required to solve the LP relaxation at the root node, (5) time spent on GRASP, and (6) average time spent at each node. Tables 5 to 8 present identical information for some of the more difficult data sets.

Generally speaking, problems with loose time windows tended to be more difficult to solve. For the larger instances, this was evidenced by the size of the search trees associated with the data sets lpltw and spltw compared to the others. In addition, all instances not solved belong to these two groups. Looking at Table 6, for example, we see that no solutions were found for three instances. In all cases, the computations terminated abruptly within 2 h due to memory limitations.

Results for 50-job, 2-machine instances in which the jobs are evenly divided into two priority classes are summarized in Tables 9 and 10. The optimal values in this case are the weighted number of jobs scheduled, where the weights reflect the smallest possible values that ensure strict priority enforcement. For the priority 1 jobs, the weights are 26; for priority 2 jobs the weights are 1. As can be seen, the results are similar to those obtained for the single priority data sets. This was to be expected because the difference between the two models is only in the definition of the cost coefficients.

Table 11 summarizes the computational results for all single priority instances investigated. Problems with up to 100 jobs and 2 machines were solved to optimality, most within a

Table 8. Computational times for 100-job, 2-machine instances (s).

Name	Total time	Time to best solution	Time at root	Time for GRASP	Average time per node
lpltw1	6411.25	a	126.54	a	a
lpltw2	205.63	205.63	71.96	69.98	10.9822
lpltw3	6224.2	a	129.55	a	a
lpltw4	4824.48	4824.48	125.49	1660.24	13.36
lpltw5	696.33	695.3	90.05	367.82	3.61
lpttw1	0.77	0.77	0.6	0	0.375
lpttw2	63.37	47.76	0.84	46.59	5.55
lpttw3	18.45	18.45	0.45	17.65	0.256667
lpttw4	79.93	79.93	0.75	63.58	2.70167
lpttw5	0.22	0.22	0.22	0	0.22
rand1	1.84	1.84	1.83	0	1.83
rand2	30.16	0	29.97	0	29.97
rand3	83.64	83.64	2.7	19.88	15.6225
rand4	90.38	90.38	2.14	31.42	9.974
rand5	42.94	42.94	42.75	0	42.75
spltw1	272.83	272.83	91.93	98.79	11.1575
spltw2	798.93	798.92	99.55	231.44	12.0207
spltw3	936.05	936.05	107.89	350.72	7.35071
spltw4	2779.23	2779.23	87.9	1018.89	12.58
spltw5	226.75	226.75	89.99	49.17	17.275
spttw1	0.47	0	0.45	0	0.45
spttw2	0.87	0	0.85	0	0.85
spttw3	0.51	0.51	0.5	0	0.5
spttw4	29.71	29.71	0.55	28.36	0.254
spttw5	0.28	0.28	0.28	0	0.28
Average ^b	499.01	496.82	35.35	181.12	8.59

^a Branch and price ran out of memory before producing an optimal solution.

^b Average values based only on instances that were solved.

matter of seconds or minutes. Our experience with the B&P algorithm confirmed the effectiveness of both the upper- and lower-bounding procedures to limit the size of the search trees to no more than a handful of nodes on average. For the 20-job, 2-machine data sets, all 25 instances were solved at the root node and, in all but one case (rand4), both the GRASP and the LP relaxation produced the same solutions (upper and lower bounds were equal). This observation is consistent with the results obtained by Bard, Kontorovdis, and Yu [3], who embedded a GRASP in a branch-and-cut procedure for the VRPTW.

Using GRASP to find feasible solutions at a node was instrumental in solving 18 of the 175 instances. Some additional testing showed that, without the GRASP, several of those problems would not have been solved. With regard to problem difficulty, increasing the number of machines appears to have a pronounced effect on algorithmic performance. Examining the results for the 30-job, 6-machine instances, for example, we see that 22 of 25 instances were solved to optimality while all 25 of the 50-job, 2-machine instances were solved. In addition, the size of the search tree grew much more rapidly with the number of machines than with the number of jobs. These observations can be partially explained by the fact that the solution space is somewhat

symmetric with respect to the definition of the data sets and that its size grows exponentially with the number of machines.

A second indicator of problem difficulty is the average number of columns per node. For problems with loose time windows, many similar columns may be generated before the relaxed master problem is optimized. As is the case with standard branch and bound, branch and price struggles when it is not able to find a near-optimal solution within the first several dozen nodes. For those instances in which the optimal solution was never found, no nodes were ever fathomed due to integrality; i.e., the LP solutions to the relaxed master problem were never integral.

7. SUMMARY AND CONCLUSIONS

Parallel machine scheduling with time windows is an extremely difficult combinatorial optimization problem, posing an array of challenges for the research community. For the variant addressed in this paper, the novelty was the presence of priorities, two services, and double time windows. Each of these features added a degree of complexity that required significant developmental work.

Table 9. Computational results for 50-job, 2-machine, 2-priority instances (B&P).

Name	Optimal solution	LP % gap at root	Columns generated at root	Average columns per node	Node with first integer solution	Total nodes	GRASP improves	GRASP calls
lpltw1	667	0.00	113	599.50	3	3	0	0
lpltw2	659	0.23	100	659.86	-1	7	0	2
lpltw3	668	0.11	2044	2145.43	4	7	0	2
lpltw4	667	0.00	145	145.00	-1	1	0	0
lpltw5	616	0.00	91	91.00	-1	1	0	0
lpttw1	531	0.00	22	22.00	-1	1	0	0
lpttw2	553	0.00	56	56.00	-1	1	0	0
lpttw3	580	0.00	47	47.00	-1	1	0	0
lpttw4	525	0.00	34	34.00	-1	1	0	0
lpttw5	505	0.00	39	58.50	3	3	0	0
rand1	611	0.00	102	111.00	3	3	0	0
rand2	611	0.00	68	68.00	-1	1	0	0
rand3	609	0.00	40	40.00	-1	1	0	0
rand4	611	0.00	54	62.00	2	3	1	1
rand5	559	0.00	68	68.00	-1	1	0	0
spltw1	670	0.10	1108	2062.58	23	23	0	4
spltw2	666	0.35	2035	2076.00	-1	3	0	1
spltw3	639	0.00	115	115.00	-1	1	0	0
spltw4	668	0.05	117	117.00	-1	1	0	0
spltw5	643	0.00	128	128.00	-1	1	0	0
spttw1	582	0.00	50	50.00	-1	1	0	0
spttw2	535	0.00	51	51.00	1	1	0	0
spttw3	560	0.00	53	53.00	-1	1	0	0
spttw4	612	0.00	85	85.00	1	1	0	0
spttw5	481	0.00	40	40.00	-1	1	0	0
Average	601	0.03	272.2	359.39	0.92	2.76	0.04	0.40

Table 10. Computational times for 50-job, 2-machine, 2-priority instances (s).

Name	Total time	Time to best solution	Time at root	Time for GRASP	Average time per node
lpltw1	22.58	22.58	1.24	0	11.23
lpltw2	88.97	0	1.5	6.06	11.79
lpltw3	227.26	227.26	185.92	8.23	29.40
lpltw4	3.07	0	3.05	0	3.05
lpltw5	0.9	0	0.9	0	0.90
lpttw1	0.1	0	0.1	0	0.10
lpttw2	0.2	0	0.19	0	0.19
lpttw3	0.16	0	0.16	0	0.16
lpttw4	0.14	0	0.13	0	0.13
lpttw5	0.38	0.38	0.17	0	0.18
rand1	0.58	0.58	0.45	0	0.28
rand2	0.3	0	0.3	0	0.30
rand3	0.25	0	0.25	0	0.25
rand4	3.3	3.14	0.31	2.76	0.17
rand5	0.17	0	0.17	0	0.17
spltw1	181.67	181.67	38.32	13.86	12.08
spltw2	107.67	0	92.8	3.79	33.20
spltw3	0.9	0	0.89	0	0.89
spltw4	1.94	0	1.92	0	1.92
spltw5	1.38	0	1.37	0	1.37
spttw1	0.25	0	0.25	0	0.25
spttw2	0.24	0.24	0.24	0	0.24
spttw3	0.25	0	0.24	0	0.24
spttw4	0.35	0.35	0.34	0	0.34
spttw5	0.15	0	0.14	0	0.14
Average	25.73	17.45	13.25	1.39	4.36

Table 11. Summary of branch-and-price computational results.

Problem size		Average ^a optimal solution	Average ^a LP-IP gap	Problems solved to optimum (of 25)	Problems solved at root	Problems solved later by GRASP	Average ^a total nodes	Average ^a columns per node	Average ^a CPU time (s)
Jobs	Machines								
20	2	14.72	0.32	25	25	0	1.00	30.88	0.05
20	6	17.20	1.35	25	21	0	6.28	54.09	0.25
30	2	22.28	0.55	25	24	0	1.08	49.00	0.26
30	6	26.09	1.60	22	7	9	310.55	118.80	48.52
50	2	36.80	0.40	25	22	0	1.64	320.10	8.24
50	3	41.55	0.90	20	4	6	249.20	1540.42	377.53
100	2	7.80	0.31	23	8	3	40.82	1185.39	499.01

^a Average values based only on instances that were solved.

Initial efforts were aimed at trying to solve 20-job, 6-machine instances with CPLEX. Lack of success led to the design of a B&P algorithm based on Dantzig–Wolfe decomposition. When coupled with a GRASP for obtaining lower bounds, instances with up to 100 jobs and 2 machines were solved to optimality, often within a few dozen seconds and within the first few nodes of the search tree. For the more difficult 100-job instances in which the algorithm failed to converge after a large number of nodes were explored, lack of sufficient memory was always the reason. In all likelihood, though, no amount of memory would have changed these results.

Relatively speaking, we found that increasing the number of machines caused the problem to become much harder than increasing the number of jobs. Because the algorithm stalled when it was not able to solve a problem early in the tree, the design of more effective branching schemes might represent one area of future research. Only time window and SOS branching were considered here. Other possibilities for future research include more efficient column management procedures and the identification of cuts that could be included in the master problem.

The question remains, though, of how to find good solutions to instances arising in the TDRSS application that motivated this work. Recall that such instances include up to 400 jobs, 6 machines, and 20 priority classes. One approach would be to use a rolling horizon framework. For each priority class, the jobs would be ordered by their earliest start times, and a manageable number, say the first 30, would be scheduled with the algorithm. Given a solution, a fraction of the 30 jobs, say the first 20, would be fixed. The remaining 10 plus the next 20 would then be scheduled and the first 20 again fixed. With suitable modifications to account for the priority classes, a complete schedule could be obtained by repeating the process a sufficient number of times. Although the results would most likely be suboptimal, the length of the time windows relative to the 24-h planning horizon suggests that the loss in optimality would be more

than offset by the expected gain in computational efficiency. Difficult optimization problems often require this type of tradeoff.

In summary, the B&P algorithm combined with the GRASP was able to solve many instances that were believed to be beyond the capabilities of exact methods. The success of the approach is mainly due to the consistency with which it can generate extremely tight upper and lower bounds throughout the search tree.

REFERENCES

- [1] E.M. Arkin and E.B. Silverberg, Scheduling jobs with fixed start and end times, *Discrete Appl Math* 18 (1987), 1–8.
- [2] E. Balas, The prize collecting traveling salesman problem, *Networks* 19 (1989), 621–636.
- [3] J.F. Bard, G. Kontoravdis, and G. Yu, A branch-and-cut procedure for the vehicle routing problem with time windows, *Transport Sci* 36 (2002), 250–269.
- [4] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance, Branch-and-price: column generation for solving huge integer programs, *Oper Res* 46 (1998), 316–329.
- [5] K.I. Bouzina and H. Emmons, Interval scheduling on identical machines, *J Global Optimization* 9 (1996), 379–393.
- [6] S.E. Butt and T.M. Cavalier, A heuristic for the multiple tour maximum collection problem, *Comput Oper Res* 21 (1994), 101–111.
- [7] M.A. Chan, A. Muriel, and D. Simchi-Levi, Parallel machine scheduling, linear programming and parameter list scheduling heuristics, *Oper Res* 46 (1998), 729–741.
- [8] M. Desrochers, J. Desrosiers, and M.M. Solomon, A new optimization algorithm for the vehicle routing problem with time windows, *Oper Res* 40 (1992), 342–354.
- [9] M. Desrochers and F. Soumis, A generalized permanent labelling algorithm for the shortest path problem with time windows, *INFOR* 26 (1988), 191–212.
- [10] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis, “Time constrained routing and scheduling,” *Handbook in operations research and management science: Network routing*, Vol. 8, M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser (Editors), North-Holland, New York, 1995, pp. 35–140.

- [11] M. Dror, Note on the complexity of the shortest path models for column generation in VRPTW, *Oper Res* 42 (1994), 977–978.
- [12] V. Gabrel, Scheduling jobs within time windows on identical parallel machines: New model and algorithms, *Eur J Oper Res* 83 (1995), 320–329.
- [13] B.L. Golden, L. Levy, and R. Vohra, The orienteering problem, *Naval Res Logist Q* 34 (1987), 307–318.
- [14] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, Optimization and approximation in Deterministic Sequencing and Scheduling: A survey, *Ann Discrete Math* 5 (1979), 287–326.
- [15] E. Horowitz, S. Sahni, and S. Anderson-Freed, *Fundamentals of data structures in C*, Computer Science Press, New York, 1993.
- [16] A.W. Kolen and L.G. Kroon, On the computational complexity of (maximum) shift class scheduling, *Eur J Oper Res* 64 (1993), 138–151.
- [17] G. Laporte and S. Martello, The selective traveling salesman problem, *Discrete Appl Math* 26 (1990), 193–207.
- [18] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, Sequencing and scheduling: Algorithms and complexity, *Handbook in operations research and management science: Logistics of production and inventory*, Vol. 4, S.S. Graves, A.H.G. Rinnooy Kan, P. Zipkin (Editors), North-Holland, New York, 1993, pp. 445–522.
- [19] W.B. Powell and Z.-L. Chen, A generalized threshold algorithm for the shortest path problem with time windows, *DIMACS Ser Discrete Math Theor Comput Sci* 40 (1998), 303–318.
- [20] S. Rojanasoonthon, Parallel machine scheduling with time windows, Ph.D. dissertation, Graduate Program in Operations Research and Industrial Engineering, University of Texas, Austin, Texas, 2004.
- [21] S. Rojanasoonthon, J.F. Bard, and S.D. Reddy, Algorithms for scheduling of the tracking and data relay satellite system, *J Oper Res Soc* 54 (2003), 806–821.
- [22] L.A. Wolsey, *Integer programming*, Wiley, New York, 1998.
- [23] T.C.E. Cheng and C.C.S. Sin, A state-of-the-art review of parallel-machine scheduling research, *Eur J Oper Res* 47 (1990), 271–292.
- [24] J. Blazewicz, M. Dror and J. Weglarz, Mathematical programming formulations for machinery scheduling: A survey, *Eur J Oper Res* 51 (1991), 283–300.