SABANCI UNIVERSITY
Orhanlı-Tuzla, 34956 Istanbul, Turkey
Phone: +90 (216) 483-9500
Fax: +90 (216) 483-9550
http://www.sabanciuniv.edu

http://algopt.sabanciuniv.edu/projects
July 25, 2010

A Hybrid Shifting Bottleneck-Tabu Search Heuristic for the Job Shop Total Weighted Tardiness Problem

Kerem Bülbül
Sabancı University, Manufacturing Systems and Industrial Engineering, Orhanlı-Tuzla, 34956 Istanbul, Turkey
bulbul@sabanciuniv.edu

Abstract: In this paper, we study the job shop scheduling problem with the objective of minimizing the total weighted tardiness. We propose a hybrid shifting bottleneck - tabu search (SB-TS) algorithm by replacing the re-optimization step in the shifting bottleneck (SB) algorithm by a tabu search (TS). In terms of the shifting bottleneck heuristic, the proposed tabu search optimizes the total weighted tardiness for partial schedules in which some machines are currently assumed to have infinite capacity. In the context of tabu search, the shifting bottleneck heuristic features a long-term memory which helps to diversify the local search. We exploit this synergy to develop a state-of-the-art algorithm for the job shop total weighted tardiness problem (JS-TWT). The computational effectiveness of the algorithm is demonstrated on standard benchmark instances from the literature.

Keywords: job shop; weighted tardiness; shifting bottleneck; tabu search; preemption; transportation problem.

1. Introduction The classical job shop scheduling problem with the objective of minimizing the makespan is one of the archetypal problems in combinatorial optimization. From a practical perspective, job shops are prevalent in shops and factories which produce a large number of custom orders in a process layout. In this setting, each order visits the resources on its prespecified route at most once. The fundamental operational problem a dispatcher faces here is to decide on a processing sequence for each of the resources given the routes and processing requirements of the orders. In the classical case described as $Jm//C_{\max}$ in the common three field notation of Graham et al. (1979), the objective is to minimize the completion time of the order that is finished latest, referred to as the makespan. This objective tends to maximize throughput by minimizing idle time in the schedule. There is a vast amount of work on minimizing the makespan in a job shop, and virtually all types of algorithms developed for combinatorial optimization problems have been tried on this problem. See Jain and Meeran (1999) for a somewhat outdated but extensive review on $Jm//C_{\max}$. On the other hand, the literature on due date related objectives in a job shop is at best scarce. Such objectives are typically associated with customer satisfaction and service level in a make-to-order environment and either penalize or prohibit job completions later than a quoted due date or deadline. In this work, we study the job shop scheduling problem with the objective of minimizing the total weighed tardiness, described in detail in the following.

We consider a job shop with m machines and n jobs. The route of a job j through the job shop is described by an ordered set $M_j = \{o_{ij} | i \in \{1, \ldots, m\}\}$, where operation o_{ij} is performed on machine i for a duration of p_{ij} time units, referred to as the processing time of o_{ij}. The kth operation in M_j is represented by $o_{[k]j}$. The start and completion times of operation o_{ij} are denoted by S_{ij} and C_{ij}, respectively, where these are related by $C_{ij} = S_{ij} + p_{ij}$. The ordered set M_j specifies the operation precedence constraints of job j, and if o_{kj} appears later than o_{ij} in M_j, then $C_{kj} \geq C_{ij} + p_{kj}$ must hold in a feasible schedule. Moreover, no operation of job j may be performed earlier than its ready time $r_j \geq 0$, and we have $S_{ij} \geq r_j$, $\forall o_{ij} \in M_j$. The completion time C_j of job j refers to the completion time of the final operation of job j, i.e., $C_j = \max_{o_{ij} \in M_j} C_{ij}$. A due date d_j is associated with each job j, and we incur a penalty per unit time of w_j if job j completes processing after d_j. Thus, the objective is to minimize the total weighted tardiness $\sum_j w_j T_j$ over all jobs, where the tardiness of job j is calculated as $T_j = \max(0, C_j - d_j)$. Also note that all ready times, processing times and due dates are assumed to be integer in this paper. All machines are available continuously from time zero onward, and a machine can execute at most one operation at

a time. The set of all operations to be performed on machine i are represented by J_i. An operation must be carried out to completion once started, i.e., preemption is not allowed. Under these definitions and constraints, the non-preemptive job shop scheduling problem with the objective of minimizing the total weighted tardiness (JS-TWT) is described as $Jm/r_j/\sum_j w_j T_j$ and is formulated below:

$$\textbf{(JS-TWT)} \qquad \min \sum_{j=1}^{n} w_j T_j \tag{1}$$

s.t.

$$C_{[1]j} \geq r_j + p_{[1]j} \qquad j = 1, \ldots, n, \tag{2}$$

$$C_{[k]j} - C_{[k-1]j} \geq p_{[k]j} \qquad j = 1, \ldots, n,\ k = 2, \ldots, |M_j|, \tag{3}$$

$$C_{[|M_j|]j} - T_j \leq d_j \qquad j = 1, \ldots, n, \tag{4}$$

$$C_{ij} - C_{ik} \geq p_{ij} \text{ or } C_{ik} - C_{ij} \geq p_{ik} \qquad \forall o_{ij}, o_{kj} \in J_i, \tag{5}$$

$$T_j \geq 0 \qquad j = 1, \ldots, n. \tag{6}$$

The constraints (2) and (3), referred to as the operation precedence constraints, ensure that the first operation of job j, $j = 1, \ldots, n$, starts no earlier than its ready time r_j, and job j, $j = 1, \ldots, n$, follows its processing sequence $M_{[1]j}, \ldots M_{[|M_j|]j}$ through the job shop, respectively. The relationship between the completion time of the final operation of job j, $j = 1, \ldots, n$, and its due date d_j is established by constraints (4). The machine capacity constraints (5) prescribe that no two operations executed on the same machine overlap. JS-TWT is strongly $\mathcal{NP}$-hard because the strongly $\mathcal{NP}$-hard single-machine scheduling problem $1/r_j/\sum w_j T_j$ (see Lenstra et al. (1977)) may be reduced to JS-TWT by setting $m = 1$.

The literature on our problem JS-TWT is limited. To the best of our knowledge, there is a single paper by Singer and Pinedo (1998) which designs an optimal algorithm for this problem. The optimal objective values for the standard benchmark test suite in Section 3 are from this paper. In an early study, Vepsalainen (1987) compare a number of dispatch rules and conclude that their proposed dispatch rule Apparent Tardiness Cost (ATC) beats alternate rules. Pinedo and Singer (1999) present an SB heuristic for JS-TWT. However, the subproblem definition and the related optimization techniques, the re-optimization step, and the other control structures in their SB heuristic are different than those proposed in this paper. We will point out the specific differences in the relevant sections. Building on this work, Singer (2001) develops an algorithm geared toward larger instances of JS-TWT with up to 10 machines and 100 jobs, where a time-based decomposition technique is applied and the subproblem for each time window is solved by the shifting bottleneck heuristic of Pinedo and Singer (1999). The next three papers by Kreipl (2000), De Bontridder (2005), and Essafi et al. (2008) all incorporate metaheuristics in their effort to solve JS-TWT effectively. In the large step random walk of Kreipl (2000), the initial solution is obtained by applying the Shortest Processing Time (SPT) dispatch rule. A neighboring solution is defined according to the neighborhood generator of Suh (1988) which we also adopt for our use after some modifications as discussed in detail in Section 2.3. The defining property of this neighborhood is that several adjacent pairwise interchanges on different machines are carried out in one move. The algorithm of Kreipl (2000) alternates between intensification and diversification phases. For diversification purposes, only the critical path of the job with the most adverse impact on the objective function is taken into account while constructing the neighborhood of the current solution. The intensification phase considers all critical paths. The total weighted tardiness problem in a job shop with generalized precedence relationships among operations is solved through a tabu search algorithm by De Bontridder (2005). Similar to Kreipl (2000), the neighborhood of a given solution in this work consists of adjacent pairwise interchanges of operations. These adjacent pairwise interchanges are identified based on the solution of a maximum flow problem that calculates the optimal operation start and completion times given the operation execution sequences of the machines. In a recent paper, Essafi et al. (2008) apply an iterated local search algorithm to improve the quality of the chromosomes in their genetic algorithm. Swapping the execution order of two adjacent operations on a critical path for any job leads to a new solution in the neighborhood. Combined with a design of experiments approach for tuning the parameter values in their algorithms, these authors develop a powerful method for solving JS-TWT effectively. The algorithms by Pinedo and Singer (1999), Kreipl (2000), De Bontridder (2005), and Essafi et al. (2008) form the state-of-the-art for JS-TWT, and we benchmark our proposed

algorithms against these in our computational study in Section 3.

In other related research, Mattfeld and Bierwirth (2004) propose a genetic algorithm for various tardiness related objectives. Armentano and Scrich (2000) suggest a tabu search algorithm for the unweighted version of JS-TWT, where a dispatch rule yields the initial solution that is improved by applying a tabu search method. The neighborhood of the current solution is formed by applying a single adjacent pairwise interchange to a pair of operations that are located on the critical path of a tardy job. In addition, there is a stream of literature on so-called *complex* job shops that incorporate features such as parallel machines, batching machines, sequence-dependent setups, re-entrant product flows, etc., in a job shop setting with the objective of minimizing the total weighted tardiness of customer orders. For instance, see Mason et al. (2002).

Before proceeding with the details of our solution approach, we briefly summarize our contributions in this paper. We propose a hybrid algorithm that substitutes the classical re-optimization step in the SB framework by tabu search. One of our main insights is that embedding a local search algorithm into the SB heuristic provides a powerful tool for diversifying any local search algorithm. In the terminology of the tabu search, the tree control structure in the SB algorithm, discussed in Section 2.4, may be regarded as a long-term memory that helps us to guide the search into previously unexplored parts of the feasible region. From the perspective of the SB heuristic, we apply tabu search both to feasible full schedules for JS-TWT and to partial schedules in which some machines are currently assumed to have infinite capacity. This is a relatively unexplored idea in SB algorithms. One excellent implementation of this idea is supplied by Balas and Vazacopoulos (1998) for the classical job shop scheduling problem. Furthermore, we underline that there is no random element incorporated into our algorithms, and by simply putting more effort into the tree search in the SB heuristic, we can ensure that progressively improved solutions are constructed. In our opinion, combined with the repeatability of results, this is an important edge of our approach over existing algorithms for JS-TWT built on random operators, e.g., those by Kreipl (2000), De Bontridder (2005), and Essafi et al. (2008).

Another significant contribution of our work is a new approach for solving a generalized single-machine weighted tardiness problem that arises as a subproblem in the SB heuristic. The original subproblem definition is due to Pinedo and Singer (1999) and Pinedo (2008), but our proposed solution method for this problem derives from our earlier work on the single-machine earliness/tardiness (E/T) scheduling problem in Bulbul et al. (2007) and yields both a lower bound and a feasible solution for the subproblem.

As pointed out earlier in this section, all local search algorithms designed for JS-TWT up until now base their neighborhood definitions on a single adjacent pairwise interchange, except for Kreipl (2000) who perform up to three adjacent pairwise interchanges, each on a different machine. In this paper, we adapt an insertion-type neighborhood definition by Balas and Vazacopoulos (1998), originally developed for $Jm//C_{\max}$, to JS-TWT. A move in this neighborhood may reverse several disjunctive arcs simultaneously (see Section 2.3) and generalizes adjacent pairwise interchanges. We argue that this neighborhood definition and Kreipl's neighborhood definition have complementary properties. We are not aware of such a dual neighborhood definition in the context of our problem, and the computational results attest to the advantages.

In Sections 2.1 through 2.4, we explain the ingredients of our state-of-the-art SB heuristic for JS-TWT in detail. Our computational results are presented in Section 3 followed by our concluding remarks in Section 4.

2. Solution Approach The basic framework of our solution approach for JS-TWT is defined by the shifting bottleneck heuristic originally proposed by Adams et al. (1988) for $Jm//C_{\max}$. The SB algorithm is an iterative machine-based decomposition technique. The fundamental idea behind this general scheduling heuristic is that the overall quality of a schedule is determined by the schedules of a limited number of machines or workcenters. Thus, the primary effort in this algorithm is spent in prioritizing the machines which dictates the order in which they are scheduled and then scheduling each machine one by one in this order. The essential ingredients of any SB algorithm are a disjunctive graph representation of the problem, a subproblem formulation that helps us both to identify and schedule the machines in some order defined by an appropriate machine criticality measure, and a rescheduling step that re-evaluates and modifies previous scheduling decisions. In the following sections, we introduce and examine each

of these steps in detail in order to design an effective SB method for JS-TWT.

2.1 Disjunctive Graph Representation The SB heuristic is a machine-based decomposition approach, and the disjunctive graph establishes the relationship between the overall problem and the subproblems. The disjunctive graph representation was first proposed by Roy and Sussman (1964) and is illustrated in Figure 1. In this figure, the job routes are given by $M_1 = \{o_{11}, o_{21}\}$, $M_2 = \{o_{22}, o_{12}, o_{32}\}$, and $M_3 = \{o_{13}, o_{23}, o_{33}\}$.

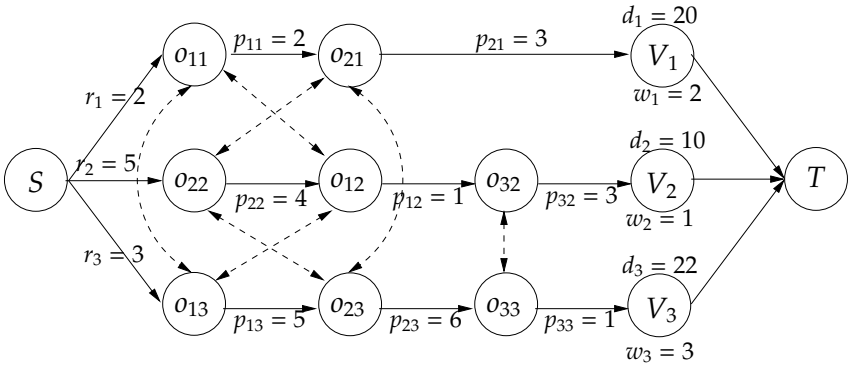
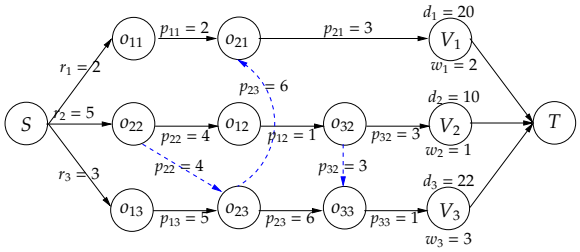


Figure 1: Disjunctive graph representation for JS-TWT.

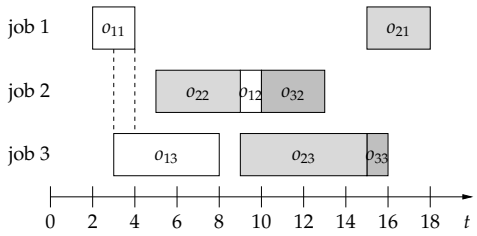
In the disjunctive graph $G(N, A)$, the node set N is given by $N = \{S, T\} \bigcup (\cup_{j=1}^{n} M_j) \bigcup (\cup_{j=1}^{n} \{V_j\})$, where S and T are the dummy source and sink nodes which mark the start of operations in the job shop at time zero and the completion of all operations, respectively. Node V_j, $j = 1, \ldots, n$, is referred to as the terminal node for job j and is associated with the completion of all operations of job j. In addition to these, we have one node per operation $o_{ij} \in \cup_{j=1}^{n} M_j$. The arc set $A = A_C \cup A_D$ is composed of two types of arcs, where the set of arcs $A_C = (\cup_{j=1}^{n} \{(S, o_{[1]j})\}) \bigcup (\cup_{j=1}^{n} \{(o_{[k-1]j}, o_{[k]j}) | k = 2, \ldots, |\ M_j\ |\}) \bigcup (\cup_{j=1}^{n} \{(o_{[|M_j|]j}, V_j)\}) \bigcup (\cup_{j=1}^{n} \{(V_j, T)\}$, and $A_D = \cup_{i=1}^{m} \{(o_{ij}, o_{ik}) | o_{ij}, o_{ik} \in J_i, o_{ij} \neq o_{ik}\}$ are referred to as the conjunctive and disjunctive arcs, respectively. The conjunctive arcs help us model the operation precedence constraints (2)-(3) and are depicted by solid lines in Figure 1. The disjunctive arcs correspond to the machine capacity constraints (5) and are illustrated by dashed lines in Figure 1. All arcs originating at an operation node o_{ij} are of length p_{ij}, and the length of an arc $(S, o_{[1]j})$ emanating from S is set to r_j in order to account for the ready time constraints (2).

Any semi-active feasible schedule for JS-TWT is associated with a graph $G'(N, A_C \cup A_D^S)$ obtained from the disjunctive graph G, where we add exactly one arc in each pair of disjunctive arcs (o_{ij}, o_{ik}), (o_{ik}, o_{ij}) to a set A_D^S while discarding the other one. This is equivalent to deciding whether o_{ij} precedes o_{ik} on machine i or vice versa and is also referred to as fixing (or orienting) a pair of disjunctive arcs. We also assume that all redundant disjunctive arcs implied by transitivity relationships are removed from A_D^S. Thus, one conjunctive arc and at most one disjunctive arc originate at an operation node o_{ij} in G'. The graph G' is associated with a feasible semi-active schedule for JS-TWT if and only if it is acyclic. Moreover, the operation and job completion times are determined by the longest paths in G', where $LP^{G'}(n_1, n_2)$ represents the length of the longest path from node n_1 to node n_2 in G'. Then, the completion time $C_{ij}(G')$ of operation $o_{ij} \in (\cup_{j=1}^{n} M_j)$ is given by $C_{ij}(G') = LP^{G'}(S, o_{ij}) + p_{ij}$. Similarly, the completion time of job j in G' is computed as $C_j(G') = LP^{G'}(S, V_j)$, and the objective value corresponding to the schedule associated with G' is computed as $\sum_{j=1}^{n} w_j \max(0, C_j(G') - d_j)$. The longest paths in G' from S to every other node in the network are computed by first topologically sorting the nodes and then processing them one by one in this order. This algorithm runs in $O(nm)$ time because all nodes in G' have at most two outgoing arcs, except for the dummy source node S with n outgoing arcs. Finally, note that if A_D^S is missing both arcs in one or several disjunctive pairs then G' corresponds to a relaxation in which several operations may be performed in parallel on a machine. For instance, in Figure 2(a) machines 2 and 3 are already scheduled and the corresponding disjunctive arcs are fixed as illustrated by the thick dashed blue arcs. However, all disjunctive arcs between the operations on machine 1 are absent from Figure 2(a), and in the schedule corresponding to this figure operations o_{11}, o_{12}, o_{13} may be simultaneously executed on machine 1 if necessary. The associated Gantt chart in Figure 2(b) reveals that the operations o_{11} and o_{13}

overlap during the time interval [3, 4] on machine 1.



(a) Disjunctive arcs (o_{22}, o_{23}), (o_{23}, o_{21}), and (o_{32}, o_{33}) are fixed. The others are deleted.



(b) Corresponding Gantt chart.

Figure 2: Disjunctive graph representation for JS-TWT.

The SB heuristic starts with no machine scheduled, i.e., we initially have $G'(N, A_C \cup \emptyset)$. In other words, all machines are assumed to have infinite capacity and may perform as many operations as desired simultaneously. Then, at each iteration of the SB heuristic we identify a currently unscheduled machine that has the most adverse effect on the objective function in some sense by solving an appropriate single-machine subproblem. For this machine, the sequence of operations is generated and the corresponding arcs are inserted into A_D^S. These steps are repeated until all machines are scheduled and G' corresponds to a semi-active schedule. The disjunctive graph has several roles in this process. First, given any selection $A_D^S \subseteq A_D$ of disjunctive arcs we determine the earliest start and completion times of the operations in the single-machine subproblems by the head and tail calculations in G'. As a byproduct, we also obtain the objective value associated with the schedule corresponding to G'. Second, if we detect a cycle in G' we conclude that the current selection of disjunctive arcs A_D^S is not feasible.

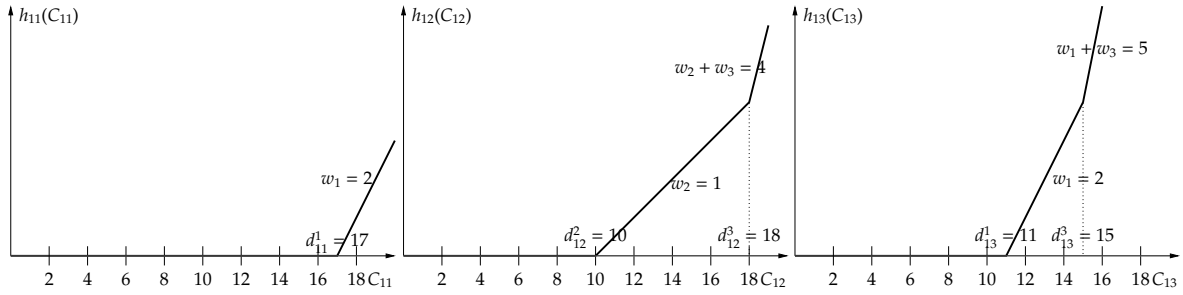
2.2 Single-Machine Subproblems A fundamental issue in any SB algorithm is to define an appropriate single-machine subproblem. Initially, the capacity constraints of all machines are relaxed by removing all disjunctive arcs. Then, the algorithm proceeds by scheduling one machine at each iteration until all machines are scheduled. The major issue at hand here is the order in which the machines are scheduled. It has been observed by many authors that the ultimate quality of the solution depends on this order to a large extent. For instance, see Aytug et al. (2002). The basic rationale of the SB framework mandates that given the set of currently scheduled machines $\mathcal{M}^S \subset \mathcal{M}$, where $\mathcal{M}$ denotes the set of all machines, we evaluate the impact of scheduling each unscheduled machine $i \in (\mathcal{M} \setminus \mathcal{M}^S)$ on the overall objective. We designate the machine deemed most critical according to some machine criticality measure as the next bottleneck machine. The common school of thought is that deferring the scheduling decisions on the current bottleneck machine any further would ultimately degrade the objective even more significantly. Thus, the objective of the single-machine subproblem is to capture the effect of scheduling a currently unscheduled machine on the overall objective function accurately. In our SB algorithm, the machine criticality measure is the objective value of the subproblem developed and discussed in detail in the sequel. At each iteration of our SB heuristic, one subproblem is set up and solved per unscheduled machine $i \in (\mathcal{M} \setminus \mathcal{M}^S)$, and the machine with the largest subproblem objective value is specified as the current bottleneck machine i^b. Then, i^b is added to $\mathcal{M}^S$ and the required disjunctive arcs for i^b are inserted into A_D^S before proceeding with the next iteration of the SB heuristic. The interested reader is referred to Aytug et al. (2002) for alternate machine criticality measures.

In developing the subproblem in the shifting bottleneck heuristic for JS-TWT, we follow the presentation in the well-known book by Pinedo (2008) in Section 7.3 and propose an effective solution method for the resulting generalized single-machine weighted tardiness problem. Assume that a subset of the machines $\mathcal{M}^S$ has already been scheduled at the start of some iteration, and the corresponding job completion times $C_j(G'(\mathcal{M}^S))$, $j = 1, \ldots, n$, are available through the longest path calculations on a graph $G'(\mathcal{M}^S) = G'(N, A_C \cup A_D^S(\mathcal{M}^S))$, where the set of disjunctive arcs $A_D^S(\mathcal{M}^S)$ is constructed according to the schedules of the machines in $\mathcal{M}^S$. Our goal is to set up a single-machine subproblem for each machine $i \in (\mathcal{M} \setminus \mathcal{M}^S)$ that computes a measure of criticality and an associated schedule for this machine if it were to be scheduled next. Clearly, the overall objective value does not decrease if the disjunctive arcs for a newly scheduled machine are inserted into $G'(\mathcal{M}^S)$. Thus, while solving the single-machine subproblems we would like to refrain from increasing the job completion times any further. To this end, we observe

that if an operation o_{ij} to be performed on machine i is not completed by a local due date d_{ij}^k, then we delay the completion of job k at a cost of w_k per unit time. The local due date d_{ij}^k depends on the longest path from o_{ij} to V_k in $G'(\mathcal{M}^S)$ and is determined as $d_{ij}^k = \max(d_k, C_k(G'(\mathcal{M}^S))) - LP^{G'(\mathcal{M}^S)}(o_{ij}, V_k) + p_{ij}$, if there is a path from o_{ij} to V_k in $G'(\mathcal{M}^S)$. Otherwise, we set $d_{ij}^k = \infty$. Consequently, the objective function of the subproblem for machine i in the shifting bottleneck heuristic is given by

$$\sum_{o_{ij} \in J_i} h_{ij}(C_{ij}) = \sum_{o_{ij} \in J_i} \sum_{k=1}^{n} w_k \max(0, C_{ij} - d_{ij}^k), \tag{7}$$

where C_{ij} is the completion time of operation o_{ij} in the subproblem, and $h_{ij}(C_{ij})$ is the associated cost function. We observe that $h_{ij}(C_{ij}) = \sum_{k=1}^{n} w_k \max(0, C_{ij} - d_{ij}^k)$ is the sum of n piecewise linear convex cost functions which implies that it is piecewise linear and convex. For instance, in Figure 2(a) machines 2 and 3 are already scheduled. The length of the longest paths from S to V_j, $j = 1, \ldots, 3$, are computed as 18, 13, and 16, respectively. Based on these values, the cost functions for the subproblem of machine 1 are calculated and depicted in Figure 3.



(a) For operation o_{11}. (b) For operation o_{12}. (c) For operation o_{13}.

Figure 3: The subproblem cost functions for the operations on machine 1. See Figure 2(a).

The analysis in this section allows us to formulate the single-machine subproblem of machine $i \in (\mathcal{M} \setminus \mathcal{M}^S)$ in the SB heuristic as a generalized single-machine weighted tardiness problem $1/r_j/\sum_j h_j(C_j)$, where the ready time of job j on machine i is given by the length of the longest path from S to o_{ij} in $G'(\mathcal{M}^S)$. For the subproblem of machine 1 in Figure 2(a), the ready times of the operations o_{11}, o_{12}, and o_{13} are determined as 2, 9, and 3, respectively. This problem is a generalization of the strongly $\mathcal{NP}$-hard single-machine weighted tardiness problem $1/r_j/\sum w_j T_j$(Lenstra et al. (1977)). Therefore, Pinedo (2008) proposes to solve $1/r_j/\sum_j h_j(C_j)$ by a generalization of the ATC dispatching rule due to Vepsalainen (1987). Note that Pinedo and Singer (1999) develop the single-machine subproblem in their shifting bottleneck algorithm for JS-TWT by taking a slightly different perspective; however, their subproblem solution approach is also based on the ATC rule. In contrast, we adapt the algorithms by Bulbul et al. (2007) originally developed for the single-machine weighted E/T scheduling problem to our generalized single-machine weighted tardiness problem.

Bulbul et al. (2007) propose a two-step heuristic in order to solve the single-machine weighted E/T scheduling problem $1/r_j/\sum_j(\epsilon_j E_j + w_j T_j)$, where E_j stands for the earliness of job j and ϵ_j is the corresponding earliness cost per unit time. In the first step, each job j is divided into p_j unit jobs, and these unit jobs are assigned over an appropriate planning horizon H by solving a transportation problem TR as defined below:

$$\textbf{(TR)} \qquad \min \sum_{j} \sum_{\substack{t \in H \\ t \geq r_j + 1}} c_{jt} X_{jt} \tag{8}$$

$$\sum_{\substack{t \in H \\ t \geq r_j + 1}} X_{jt} = p_j \qquad \forall j, \tag{9}$$

$$\sum_{\substack{j \\ t \geq r_j + 1}} X_{jt} \leq 1 \qquad \forall t \in H, \tag{10}$$

$$X_{jt} \geq 0 \qquad \forall j,\ \forall t \in H,\ t \geq r_j + 1, \tag{11}$$

where X_{jt} is set to 1 if a unit job of job j is processed in the interval $(t-1, t]$ at a cost of c_{jt}, and 0 otherwise. Moreover, if the cost coefficients c_{jt} are chosen carefully, then the optimal objective value of TR provides a tight lower bound on the optimal objective value of the original problem. Clearly, the schedule obtained from the optimal solution of this transportation problem incorporates preemptions; however, the authors observe that more expensive jobs are scheduled more or less contiguously and close to their positions in the optimal non-preemptive schedule while inexpensive jobs are preempted more frequently. Thus, in the second step the information provided in the optimal solution of this preemptive relaxation is exploited in devising several primal heuristics with small optimality gaps for the original non-preemptive problem.

The success of the approach outlined above relies essentially on the cost coefficients. The key to identifying a set of valid cost coefficients is to ensure that the cost of a non-preemptive schedule in the transportation problem is no larger than that in the original non-preemptive problem. This property does immediately lead to the result that the optimal objective value of TR is a lower bound on that of the original non-preemptive problem because the set of all feasible non-preemptive schedules is a subset of the set of all preemptive schedules. Bulbul et al. (2007) propose the cost coefficients below for $1/r_j/\sum_j(\epsilon_j E_j + w_j T_j)$:

$$c_{jt} = \begin{cases} \frac{\epsilon_j}{p_j}\left[(d_j - \frac{p_j}{2}) - (t - \frac{1}{2})\right], & \text{if } t \leq d_j, \text{ and} \\ \frac{w_j}{p_j}\left[(t - \frac{1}{2}) - (d_j - \frac{p_j}{2})\right], & \text{if } t > d_j. \end{cases} \tag{12}$$

We generalize the cost coefficients in (12) for our problem, where c_{ijt} stands for the cost of processing one unit job of operation o_{ij} on machine i during the time interval $(t-1, t]$:

$$c_{ijt} = \sum_{\substack{k=1 \\ t > d_{ij}^k}}^{n} \frac{w_k}{p_{ij}}\left[(t - \frac{1}{2}) - (d_{ij}^k - \frac{p_{ij}}{2})\right]. \tag{13}$$

Our single machine subproblems $1/r_j/\sum_j h_j(C_j)$ in the SB heuristic are regular, i.e., the objective is non-decreasing in the completion times. This implies that no job will ever finish later than $\max_j r_j + P$, where P denotes the sum of the operation processing times on the associated machine, in an optimal non-preemptive solution. Therefore, it suffices to define the planning horizon H as $H = \{k | k \in \mathbb{Z},\ \min_j r_j + 1 \leq k \leq \max_j r_j + P\}$ while solving the lower bounding problem TR. Next, we show that the optimal objective value of TR yields a valid lower bound for $1/r_j/\sum_j h_j(C_j)$ with the cost coefficients given in (13). The structure of the proof follows that of Theorem 2.2 in Bulbul et al. (2007). For brevity of notation, we omit the machine index i in the derivations.

THEOREM 2.1 *For an instance of $1/r_j/\sum_j h_j(C_j)$ with n operations, where $h_j(C_j) = \sum_{k=1}^{n} w_k \max(0, C_j - d_j^k)$, the optimal objective value z_{TR}^* of the transportation problem (8)-(11) with the cost coefficients $c_{jt} = \sum_{\substack{k=1 \\ t > d_j^k}}^{n} \frac{w_k}{p_j}\left[(t - \frac{1}{2}) - (d_j^k - \frac{p_j}{2})\right]$ for $j = 1, \ldots, n$, $t \in H$, solved over a planning horizon $H = \{k | k \in \mathbb{Z},\ \min_j r_j + 1 \leq k \leq \max_j r_j + P\}$ provides a lower bound on the optimal objective value z^* of $1/r_j/\sum_j h_j(C_j)$.*

PROOF. Any non-preemptive optimal schedule for $1/r_j/\sum_j h_j(C_j)$ is also feasible for TR if each job is divided into p_j consecutive unit jobs. The proof is then completed by showing that any non-preemptive optimal schedule incurs no larger cost in TR than that in the original non-preemptive problem.

In the following analysis, we investigate the cost incurred by any job j in a non-preemptive optimal schedule. A job j which completes at time C_j incurs a cost z_j^k in TR with respect to each of the due dates d_j^k, $k = 1, \ldots, n$. If $C_j \leq d_j^k$, then $z_j^k = 0$. Otherwise, we need to distinguish between two cases. If $C_j \geq d_j^k + p_j$, then we have

$$z_j^k = \frac{w_k}{p_j} \sum_{t=C_j - p_j + 1}^{C_j} \left[(t - \frac{1}{2}) - (d_j^k - \frac{p_j}{2})\right] = w_k(C_j - d_j^k),$$

which is identical to the cost incurred by job j in the original non-preemptive problem with respect to d_j^k. On the other hand, if $p_j \geq 2$ and $C_j = d_j^k + x$, where $1 \leq x \leq p_j - 1$, then

$$z_j^k = \frac{w_k}{p_j} \sum_{t=d_j^k+1}^{d_j^k+x} \left[(t - \frac{1}{2}) - (d_j^k - \frac{p_j}{2}) \right] = w_k x \left[\frac{x + p_j}{2p_j} \right] < w_k x = w_k (C_j - d_j^k),$$

because $x < p_j$. Thus, the total cost z_j accumulated by job j in TR is

$$z_j = \sum_{k=1}^{n} z_j^k = \sum_{t=C_j-p_j+1}^{t} c_{jt} \leq \sum_{k=1}^{n} w_k \max(0, C_j - d_j^k) = h_j(C_j)$$

for all possible values of C_j in the planning horizon H. The desired result $z_{TR}^* \leq \sum_j z_j \leq z^*$ follows by summing over all jobs. □

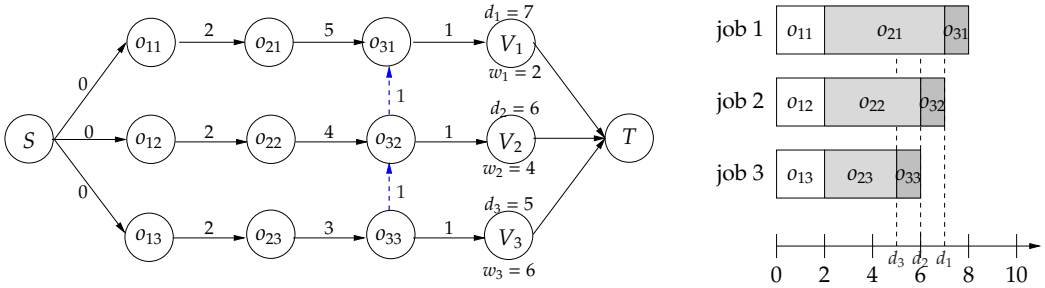
In the optimal solution of the transportation problem for machine i, jobs (operations on machine i) may be preempted at integer points in time. Thus, upon solving the transportation problem we need to apply a heuristic to its optimal solution to construct a feasible schedule for the original non-preemptive problem $1/r_j/\sum_j h_j(C_j)$. This feasible solution then dictates which disjunctive arcs are fixed for the next iteration of the SB heuristic. For this step, we directly use the heuristics proposed by Bulbul et al. (2007). Three of these heuristics rely on statistics compiled from the completion times of the unit jobs in the optimal solution of TR. In the LCT (last completion time) Heuristic, jobs are sequenced in non-decreasing order of the completion times of their last unit jobs. In the ACT (average completion time) and MCT (median completion time) Heuristics, jobs are sequenced in non-decreasing order of the average and median completion time of their unit jobs, respectively. Finally, the Switch Heuristic seeks a non-preemptive schedule by shuffling around the unit jobs while it greedily attempts to limit the increase in cost with respect to the optimal transportation solution. In all cases, once a job processing sequence is available jobs are scheduled as early as possible while observing the ready times since the single-machine subproblem is regular. The best of four possible sequences is employed to fix the disjunctive arcs between the operations on machine i.

Theoretically, the lower bound based on TR is only computed in pseudo-polynomial time because the planning horizon H depends on the sum of the operation processing times. In addition, the SB heuristic is an iterative approach which implies that this lower bounding problem is solved many times during the course of the heuristic. Thus, using this computationally expensive solution method for our subproblems needs some justification. First, we point out that the planning horizon in TR may be reduced considerably by a simple observation. Since all cost coefficients in TR are non-negative and jobs may be preempted at integer points in time, no unit job will be ever be assigned to a time period larger than $t_{\max}$, where $t_{\max}$ is the optimal objective value of $1/r_j, pmtn/C_{\max}$. This problem may be solved in $O(n \log n)$ time by sorting the operations in non-decreasing order of their ready times and then scheduling any unit job that is available as early as possible. A similar reasoning for the planning horizon is applied by Runge and Sourd (2009) in order to compute a valid lower bound for a preemptive single-machine E/T scheduling problem based on the transportation problem. Second, very large instances of the transportation problem can be solved very effectively by standard solvers and the single-machine instances derived from JS-TWT in our computational study do not have more than 20 jobs.[1] Third, in our computational study in Section 3 we demonstrate that the proposed solution method for the subproblems is viable. Forth, by scaling down the due dates, ready times, and the processing times in an instance of JS-TWT appropriately, we can decrease the time expended in solving the transportation problems in the SB heuristic significantly at the expense of losing some information for extracting a good job processing sequence from the subproblems. This idea is further developed in Section 3, and our numerical results indicate that this approach does not lead to a major loss in solution quality while it reduces the computation times.

Finally, we discuss an issue inherent in our subproblem definition. In the SB heuristic, the goal of the subproblem definition is to predict the effect of scheduling one additional machine $i \in (\mathcal{M} \setminus \mathcal{M}^S)$ on the

[1]Instances of JS-TWT with 10 machines and 15 operations per machine are already regarded as very large instances for this problem. The most famous standard benchmark problem set consists of instances with 10 machines and 10 operations per machine. For more details, see Section 3.

overall objective function. To this end, we associate a cost function $h_{ij}(C_{ij})$ with each operation o_{ij} on machine i which is an estimate of the cost of completing operation o_{ij} at time C_{ij} after the disjunctive arcs on machine i are fixed. Then, an estimate of the total increase in the overall objective after scheduling machine i is given by the sum of these individual effects $\sum_{o_{ij}\in J_i} h_{ij}(C_{ij})$. In some cases, this subproblem definition may lead to a "double-counting" as illustrated by the instance in Figure 4.

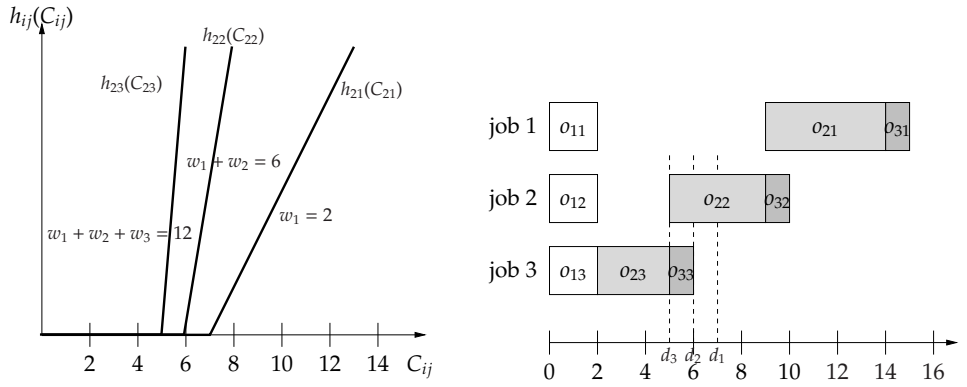


(a) Operation processing sequence on machine 3 is fixed.

(b) Current schedule.

Figure 4: Double counting in the subproblems.

In Figure 4(a), the job processing sequence on machine 3 is fixed, and the corresponding schedule is depicted in Figure 4(b) with an objective value of 12. In the subproblem for machine 2, all ready times are 2, and the cost functions are plotted in Figure 5(a). The optimal solution of this subproblem yields $C_{23} = 5$, $C_{22} = 9$, $C_{21} = 14$ with an objective value of 32. Thus, the optimal solution of the subproblem estimates that the overall objective will increase from 12 to 12+32=44 after the disjunctive arcs (o_{23}, o_{22}) and (o_{22}, o_{21}) are fixed. However, the resulting schedule in Figure 5(b) bears a total cost of only 38. Further



(a) Cost functions for the subproblem of machine 2.

(b) Schedule after the disjunctive arcs on machine 2 are fixed.

Figure 5: Double counting in the subproblems.

analysis reveals that in the subproblem we shift operation o_{22} to the right for 3 units of time ($C_{22} = 6$ in Figure 4(b)) at a cost of 6 per unit time, and operation o_{21} is pushed later for 7 units of time ($C_{21} = 7$ in Figure 4(b)) at a cost of 2 per unit time, resulting in a total cost of 32. However, if we investigate the resulting overall schedule in Figure 5(b) in detail after the disjunctive arcs on machine 2 are fixed, we conclude that the cost of pushing o_{21} later is already partially incorporated in the cost of delaying o_{22} for 3 units of time. Thus, the cost of shifting o_{21} for 3 units of time at a cost of 2 per unit time is counted twice in the subproblem objective. Summarizing, we emphasize that the operation cost functions in the subproblems may fail to take into account complicated cross effects from fixing several disjunctive arcs simultaneously. We do not have an immediate remedy for this issue and leave it as a future research direction. However, our numerical results in Section 3 attest to the reasonable accuracy of the bottleneck information and the job processing sequences provided by our subproblems.

There is one additional complication that may arise while solving the subproblems in any SB heuristic. Fixing the disjunctive arcs according to the job processing sequences of the scheduled machines may introduce directed paths in the disjunctive graph between two operations on a machine that is yet to be

scheduled. Such paths impose start-to-start time lags between two operations on the same machine, and ideally they have to be taken into account while solving the single-machine subproblems. These so-called delayed precedence constraints (DPCs) have been examined in detail by several researchers, mostly in the context of the SB algorithms developed for $Jm//C_{\max}$. For instance, see Dauzere-Peres and Lasserre (1993) and Balas et al. (1995). In this paper, our subproblem definition generalizes the strongly $\mathcal{NP}$-hard single-machine weighted tardiness problem, and we are not aware of any previously existing good algorithm for its solution, even in the absence of DPCs. Thus, we solve the subproblems without accounting for the DPCs, and then check whether the solution provided causes any infeasibility. This task is accomplished by checking for directed cycles while updating the disjunctive graph $G'(\mathcal{M}^S)$ according to the operation sequence of the latest bottleneck machine. If necessary, feasibility is restored by applying local changes to the job processing sequence on the bottleneck machine. Moreover, in our computational study we observe that only a few DPCs have to be fixed per instance solved which further justifies our approach.

2.3 Rescheduling by Tabu Search The last fundamental component of an SB heuristic is rescheduling which completes one full iteration of the algorithm. The goal of rescheduling is to re-optimize the schedules of the previously scheduled machines given the decisions for the current bottleneck machine. It is widely observed that the performance of an SB algorithm degrades considerably if the rescheduling step is omitted. For instance, see Demirkol et al. (1997). In classical SB algorithms, such as that in Pinedo and Singer (1999), rescheduling is accomplished by removing each machine $i \in (\mathcal{M}^S \setminus \{i^b\})$, where i^b is the current bottleneck machine, from the set of scheduled machines $\mathcal{M}^S$, and then updating the job processing sequence on this machine before adding it back to $\mathcal{M}^S$. To this end, all disjunctive arcs associated with machine i are first removed from the disjunctive graph $G'(\mathcal{M}^S)$, and a single-machine subproblem is defined for machine i in the usual way. Then, the disjunctive arcs for machine i are inserted back into the disjunctive graph as prescribed by the solution of the subproblem. Generally, SB algorithms perform several full cycles of rescheduling until no further improvement is achieved in the overall objective function.

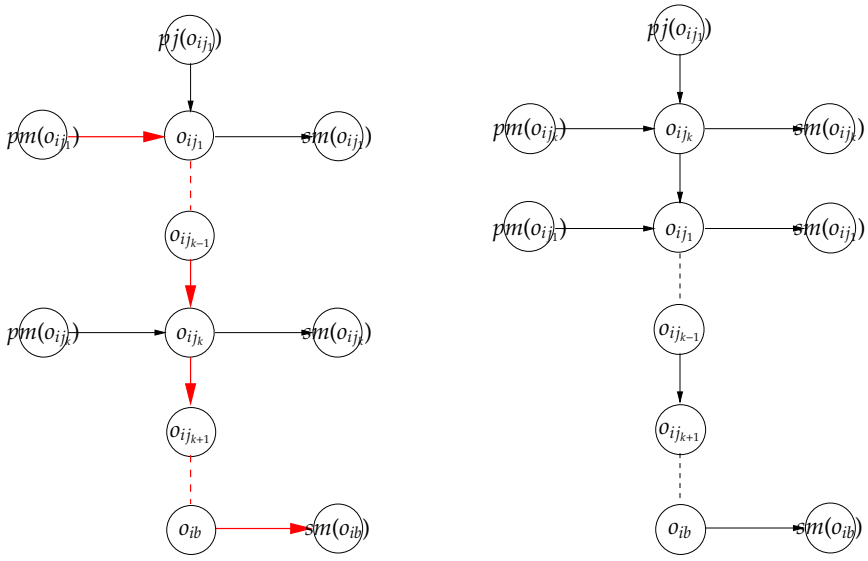
The re-optimization step of the shifting bottleneck procedure may be regarded as a local search algorithm, where the neighborhood is defined by the set of all schedules that may be obtained by changing the job processing sequence on one machine only as discussed by Balas and Vazacopoulos (1998). Intrinsically, all local search algorithms visit one or several local optima on their trajectory, and their ultimate success depends crucially on their ability to escape from the neighborhood of the current local optimum with the hope of identifying a more promising part of the feasible region. This is often referred to as *diversification* while searching for the best solution in the current neighborhood is known as *intensification*. For diversification purposes, a powerful strategy and a recent trend in heuristic optimization is to combine several neighborhoods. If the diversification procedures in place do not allow us to escape from the region around the current local optimum given the current neighborhood definition, then switching to an alternate neighborhood definition may just achieve this goal. Motivated by this observation, and following suit with Balas and Vazacopoulos (1998) we replace the classical rescheduling step in the SB heuristic discussed in the preceding paragraph by a local search algorithm. However, while Balas and Vazacopoulos (1998) employ a guided local search algorithm based on the concept of neighborhood trees for diversification purposes, we instead propose a tabu search algorithm. We also note that Balas and Vazacopoulos (1998) design a shifting bottleneck algorithm for $Jm//C_{\max}$ while we solve JS-TWT.

From a practical point of view, JS-TWT is a substantially harder problem to solve compared to the classical job shop scheduling problem $Jm//C_{\max}$. However, in both problems the concept of a critical path plays a fundamental role. In $Jm//C_{\max}$, the objective is to minimize the length of the longest path from a dummy source node S to a dummy sink node T, while in JS-TWT the objective is a function of n critical paths from S to $V_j, j = 1, \ldots, n$. (See Figure 1.) Therefore, local search algorithms or metaheuristics designed for JS-TWT generally rely on neighborhoods originally proposed for $Jm//C_{\max}$ as pointed out in Section 1 while they take the necessary provisions to deal with the dependence of the objective on several critical paths with varying degrees of importance. The local search component based on tabu search incorporated into our SB algorithm features two contributions compared to the existing literature. First, we adapt a neighborhood generation mechanism proposed by Balas and Vazacopoulos (1998) for $Jm//C_{\max}$ to JS-TWT. This neighborhood generator reverses the directions of one or several disjunctive

arcs on a given machine simultaneously, and thus is more general than the previous neighborhood definitions applied to JS-TWT. Up to date, all neighborhood generators employed for JS-TWT reverse the direction of a single disjunctive arc by applying an adjacent pairwise interchange to the job processing sequence of one of the machines. Second, the neighborhood generation scheme of Kreipl (2000) for JS-TWT, originally due to Suh (1988), is used together with the neighborhood described above after some improvements. Thus, a degree of diversification is directly built into our neighborhood generation mechanism. Our main motivation here is that these two neighborhood generation schemes have complementary properties. The neighborhood generator by Balas and Vazacopoulos (1998) applies a general interchange procedure to the job processing sequence of one machine only while the neighborhood by Kreipl (2000) may apply several adjacent pairwise interchanges simultaneously, each to the job processing sequence of a distinct machine.

Before we describe our neighborhood generation scheme in detail, a discussion of some of the basic properties of the critical paths in the disjunctive graph is in order. Given a set of currently scheduled machines $\mathcal{M}^S$, any longest path from S to T or V_j, $j = 1, \ldots, n$, in $G'(N, A_C \cup A_D^S)$ is composed of blocks of arcs, where we refer to an arc on any longest path as a *critical arc*. The arcs in each block either belong to the operations of the same job or the operations performed on the same machine. For instance, in Figure 2(a) the longest path from S to V_1 is given by an ordered set of arcs $\{(S, o_{22}), (o_{22}, o_{23}), (o_{23}, o_{21}), (o_{21}, V_1)\}$, where (S, o_{22}) and (o_{21}, V_1) each form a block of their own that belong to the set of conjunctive arcs pertaining to jobs 2 and 1, respectively, and the arcs $(o_{22}, o_{23}), (o_{23}, o_{21})$ in the middle block are included in the set of disjunctive arcs on machine 2. Clearly, in an ideal situation we would like the critical paths from S to V_j, $j = 1, \ldots, n$, to consist only of conjunctive arcs; this is the best we can achieve. Thus, the basic goal of a local search algorithm for JS-TWT is to modify the critical paths as to avoid the disjunctive arcs as much as possible while accounting for the different contributions of the critical paths to the objective function. In the context of $Jm//C_{\max}$, Matsuo et al. (1988) recognized that if (o_{ij}, o_{ik}) is a disjunctive arc on a critical path from S to T, then an adjacent pairwise interchange of the operations o_{ij} and o_{ik} on machine i may reduce the makespan only if either the job-predecessor of o_{ij} or the job-successor of o_{ik} is also on the critical path. In other words, if a critical path from S to T is described by $\{\ldots, (o_{kj_1}, o_{ij_1}), (o_{ij_1}, o_{ij_2}), (o_{ij_2}, o_{ij_3}), (o_{ij_3}, o_{ij_4}), (o_{ij_4}, o_{lj_4}), \ldots\}$, then an update to the sequence of operations o_{ij_1}, o_{ij_2}, o_{ij_13}, o_{ij_4} on machine i needs to involve at least one of the operations o_{ij_1} or o_{ij_4} as a necessary condition for reducing the makespan. The interested reader is referred to Balas and Vazacopoulos (1998) for a more in-depth discussion. For JS-TWT, we can apply this observation directly for identifying moves that may decrease the length of a critical path from S to V_j, $j = 1, \ldots, n$, because the rationale is the same whether we are interested in the length of the critical path from S to T or any other node in the network. Next, given these observations we introduce the neighborhoods we employ in the tabu search for reducing the total weighted tardiness in the current disjunctive graph $G'(N, A_C \cup A_D^S)$ during the re-optimization step of our SB algorithm. In our descriptions, we follow the notation in Kreipl (2000). We denote the immediate job predecessor and immediate job successor of operation o_{ij} by $pm(o_{ij})$ and $sm(o_{ij})$, respectively. That is, $(pm(o_{ij}), o_{ij})$ and $(o_{ij}, sm(o_{ij}))$ are members of the set of conjunctive arcs A^C. Furthermore, the immediate machine predecessor and the immediate machine successor of operation o_{ij} (if they exist) are represented by $pj(o_{ij})$ and $sj(o_{ij})$, respectively, i.e., A_D^S contains $(pj(o_{ij}), o_{ij})$ and $(o_{ij}, sj(o_{ij}))$.

In the *generalized interchange* (GI) neighborhood, adapted from Balas and Vazacopoulos (1998), we first identify the block structure for one critical path per job. Then, for each block of the form $\{(o_{ij_1}, o_{ij_2}), (o_{ij_2}, o_{ij_3}), \ldots, (o_{ij_{b-1}}, o_{ij_b})\}$ composed of disjunctive arcs on some machine i on the critical path from S to V_j, $j = 1, \ldots, n$, we identify one neighboring solution by moving operation $o_{ij_k}, k = 1, \ldots, b-1$, right after o_{ij_b} in the job processing sequence of machine i. This is referred to as a *forward interchange*. Similarly, a *backward interchange* is performed by moving an operation $o_{ij_k}, k = 2, \ldots, b$, right before o_{ij_1} in the job processing sequence of machine i as illustrated in Figure 6. The original disjunctive graph is depicted on the left in Figure 6(a), where the relevant part of the critical path from S to V_j is indicated by the bold red lines. The disjunctive graph G'' modified due to a backward interchange on o_{ij_1} and o_{ij_k} is in Figure 6(b). Note that the forward and backward interchanges reverse the directions of several disjunctive arcs simultaneously in the current disjunctive graph G', and they always involve at least one of the operations o_{ij_1} or o_{ij_b}, where $pm(o_{ij_1})$ and $sm(o_{ij_b})$ are also located on the critical path. Balas and Vazacopoulos (1998) extend the necessary condition for reducing the makespan for adjacent pairwise interchanges stated in the preceding paragraph to the GI neighborhood; and thus, both forward and backward interchanges fulfill the necessary condition for reducing the length of a critical path from



(a) Current disjunctive graph G'. (b) Modified disjunctive graph G''.

Figure 6: Backward interchange in the GI neighborhood.

S to V_j, $j = 1, \ldots, n$. A forward or backward interchange corresponds to a feasible neighboring solution given the set of currently scheduled machines $\mathcal{M}^S$ if and only if the modified disjunctive graph is acyclic. Note that the number of forward and backward interchanges for a block of size b is $2(b-1)$, and updating and checking for cycles for each of these interchanges would be computationally costly in an iterative local search algorithm. Therefore, only those forward and backward interchanges that satisfy the conditions in Propositions 2.1 and 2.2 below, respectively, are included in the neighborhood of the current solution. Propositions 2.1 and 2.2 are straightforward extensions of Propositions 2.2 and 2.3 in Balas and Vazacopoulos (1998). We only provide a proof for Proposition 2.2 because the corresponding proof in Balas and Vazacopoulos (1998) is omitted.

Proposition 2.1 *If a critical path in $G'(N, A_C \cup A_D^S)$ from S to one of the terminal nodes V_j contains o_{ij_k}, o_{ib}, and $sm(o_{ib})$, then a forward interchange on o_{ij_k} and o_{ib} leads to an acyclic disjunctive graph G'' if either there is no path from $sm(o_{ij_k})$ to V_j or the following condition is satisfied:*

$$LP^{G'}(o_{ib}, V_j) \geq LP^{G'}(sm(o_{ij_k}), V_j). \tag{14}$$

Proposition 2.2 *If a critical path in $G'(N, A_C \cup A_D^S)$ from S to one of the terminal nodes V_j contains $pm(o_{ij_1})$, o_{ij_1}, and o_{ij_k}, then a backward interchange on o_{ij_1} and o_{ij_k} leads to an acyclic disjunctive graph G'' if either $pm(o_{ij_k}) = S$ or the following condition is satisfied:*

$$LP^{G'}(S, o_{ij_1}) + p_{ij_1} \geq LP^{G'}(S, pm(o_{ij_k}))) + p_{pm(o_{ij_k})}. \tag{15}$$

Proof. By contradiction. Suppose that moving o_{ij_k} before o_{ij_1} creates a cycle C. Then, C contains either (o_{ij_k}, o_{ij_1}) or $(o_{ij_k}, sm(o_{ij_k}))$. If $(o_{ij_k}, sm(o_{ij_k})) \in C$, then there must exist a path in G' from $sm(o_{ij_k})$ to o_{ij_k}, contrary to the assumption that G' is acyclic. If $(o_{ij_k}, o_{ij_1}) \in C$, then there exists a path from o_{ij_1} to $pm(o_{ij_k})$ in G'. This would imply $LP^{G'}(S, pm(o_{ij_k})) \geq LP^{G'}(S, o_{ij_1}) + p_{ij_1}$, resulting in $LP^{G'}(S, pm(o_{ij_k})) + p_{pm(o_{ij_k})} > LP^{G'}(S, o_{ij_1}) + p_{ij_1}$ which contradicts (15). □

The condition (14) is based on the lengths of the longest paths from o_{ib} and $sm(o_{ij_k})$ to V_j while the condition (15) requires the lengths of the longest paths from S to o_{ij_1} and $pm(o_{ij_k})$. The latter come for free as a byproduct of the forward pass performed in the current disjunctive graph G' from S to T in the topological order of the nodes in order to determine the operation and job completion times and the objective value associated with G'. However, we need to carry out a backward pass from T to S in G' for the tail calculations from all nodes to a terminal node. Fortunately, by keeping one distance label per terminal node on each node, this can be accomplished effectively for all terminal nodes in one backward pass through G'.

Algorithm 1: Creating the MAI neighborhood.

input : Disjunctive arc (o_{ij}, o_{ik}) is on the critical path from S to V_j. All start and completion times below refer to those in the current disjunctive graph G'.

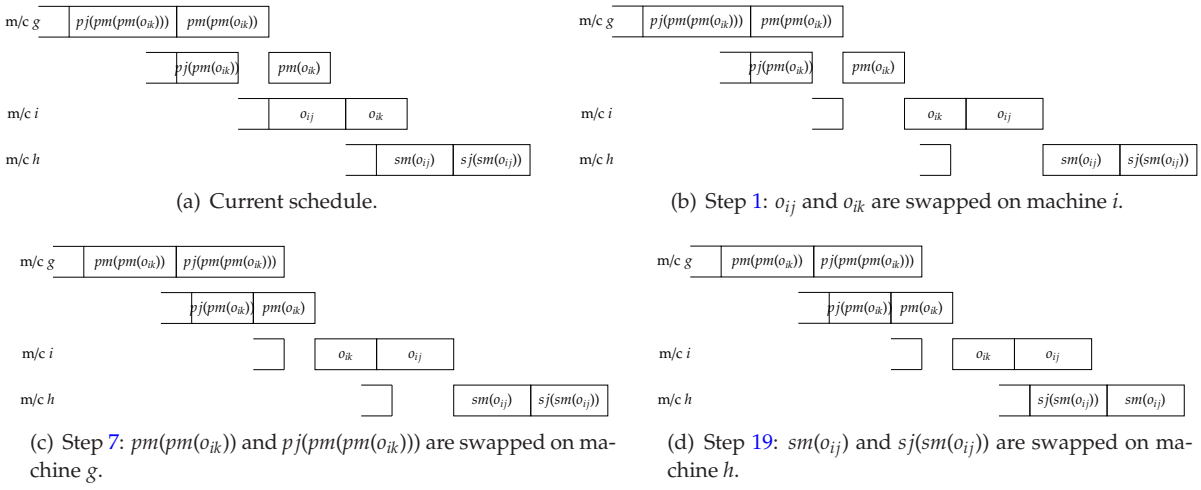
1 Swap o_{ij} and o_{ik} in the operation processing sequence of machine i;
2 $n_{int} = 1$; // Keep track of the number of adjacent pairwise interchanges.

/* Go up the predecessor chain of o_{ik} for possible reductions in the critical path length. */
3 **if** $pm(o_{ik}) \neq S$ **then** // o_{ik} is not the first operation of job k
4 **if** $C_{pm(o_{ik})} > S_{ij}$ **then** /* After swapping o_{ij} and o_{ik}, the longest path from S to o_{ik} is through $pm(o_{ik})$. */
5 **while** $(n_{int} = 1)$ **and** $(pm(o_{ik}) \neq S)$ **do** // At most one interchange in this loop
6 **if** $(pj(pm(o_{ik})) \neq \emptyset)$ **and** $(S_{pm(o_{ik})} = C_{pj(pm(o_{ik}))})$ **then** /* The longest path from S to $pm(o_{ik})$ goes through its machine predecessor. */
7 Swap $pm(o_{ik})$ and $pj(pm(o_{ik}))$ in the operation processing sequence of machine $g \neq i$;
8 $n_{int} = n_{int} + 1$;
9 **else** /* Look further up in the predecessor chain of o_{ik}. */
10 Set $pm(o_{ik}) = pm(pm(o_{ik}))$
11 **end**
12 **end**
13 **end**
14 **end**
15 Compute an estimate $\bar{C}_{ik}$ for the completion time of o_{ik} after a maximum of two interchanges performed so far;

// Check whether the starting time of $sm(o_{ij})$ is affected.
16 **if** $sm(o_{ij}) \neq V_j$ **then** // o_{ij} is not the last operation of job j.
17 **if** $S_{sm(o_{ij})} < \bar{C}_{ik} + p_{ij}$ **then** // The estimated starting time of o_{ij} is $\bar{C}_{ik}$.
18 **if** $(sj(sm(o_{ij})) \neq \emptyset)$ **and** $(C_{sm(o_{ij})} = S_{sj(sm(o_{ij}))})$ **then** /* The longest path from S to the machine successor of $sm(o_{ij})$ goes through $sm(o_{ij})$. */
19 Swap $sm(o_{ij})$ and $sj(sm(o_{ij}))$ in the operation processing sequence of machine $h \neq i, g$;
20 $n_{int} = n_{int} + 1$;
21 **end**
22 **end**
23 **end**

Identifying the members of the *multiple adjacent interchange* (MAI) neighborhood of Kreipl (2000) for JS-TWT, originally due to Suh (1988), starts similarly by identifying the block structure of one critical path per job. Then, given a disjunctive arc (o_{ij}, o_{ik}) on some critical path, regardless of whether $pm(o_{ij})$ or $sm(o_{ik})$ are also located on the critical path, the neighboring solution in the MAI neighborhood is determined by following the procedure in Algorithm 1 that is illustrated in Figure 7. We start by swapping the sequence of the operations o_{ij} and o_{ik} on machine i (*primary interchange*) in Step 1 and perform at most two additional adjacent pairwise interchanges (*secondary interchanges*) in Steps 7 and 19. We emphasize once again that these interchanges all occur on distinct machines in contrast to the interchanges in the GI neighborhood which are carried out on the same machine. Thus, these two neighborhoods complement each other and facilitate escaping from local optima during the tabu search algorithm presented later in this section. We also note that our implementation differs slightly from that by Kreipl (2000) in Steps 15 and 17 of Algorithm 1. After performing up to two adjacent pairwise interchanges in Steps 1-14, we first estimate the completion time of operation o_{ik} after these swaps in Step 15 in order to provide more accurate information in Step 17 for the last potential adjacent pairwise interchange. See Algorithm 2. In our preliminary experiments, we observed that the condition $S_{sm(o_{ij})} < \bar{C}_{ik} + p_{ij}$ in Step 17 performs better than the original condition $S_{sm(o_{ij})} < C_{ik}$ in Kreipl (2000), where the current completion time C_{ik} of operation o_{ik} in G' is used as an estimate of the completion time of o_{ij} after Steps 1-14 are completed. For the MAI neighborhood, we do not have a result similar to those in Propositions 2.1 and 2.2 for the GI

neighborhood, but moving to a neighboring solution in the MAI neighborhood never leads to a cycle in the modified disjunctive graph in our entire set of experiments in Section 3. Therefore, we conjecture that a move in the MAI neighborhood is guaranteed to produce an acyclic disjunctive graph.



(a) Current schedule.

(b) Step 1: o_{ij} and o_{ik} are swapped on machine i.

(c) Step 7: $pm(pm(o_{ik}))$ and $pj(pm(pm(o_{ik})))$ are swapped on machine g.

(d) Step 19: $sm(o_{ij})$ and $sj(sm(o_{ij}))$ are swapped on machine h.

Figure 7: The multiple interchange neighborhood described in Algorithm 1.

Algorithm 2: Calculating $\bar{C}_{ik}$ in Step 15 of Algorithm 1.

if *swap occurred in Step 7 of Algorithm 1* **then**
  $o_{cur} = o_{gk}$; // o_{cur} represents the operation currently under consideration.
else
  $o_{cur} = o_{ik}$;
end
if $pm(o_{cur}) \neq S$ **then** /* $\bar{C}_{ik}$ is initialized to the earliest starting time of o_{cur} as determined by the conjunctive arcs. */
  $\bar{C}_{ik} = C_{pm(o_{cur})}$;
else
  $\bar{C}_{ik} = r_k$;
end
repeat
  /* The current value of $\bar{C}_{ik}$ is the earliest starting time of o_{cur} as determined by its operation predecessor. The next if-statement accounts for the machine predecessor of o_{cur}. */
  if $(o_{cur} = o_{gk})$ **or** $(o_{cur} = o_{ik})$ **then** /* o_{gk} and o_{ik} are swapped with their machine predecessors. */
    $\bar{C}_{ik} = \max(\bar{C}_{ik}, C_{pj(pj(o_{cur}))})$; // $C_{pj(pj(o_{cur}))} = 0$ if $pj(pj(o_{cur}))$ does not exist.
  else
    $\bar{C}_{ik} = \max(\bar{C}_{ik}, C_{pj(o_{cur})})$; // $C_{pj(o_{cur})} = 0$ if $pj(o_{cur})$ does not exist.
  end
  $\bar{C}_{ik} = \bar{C}_{ik} + p_{o_{cur}}$; // $\bar{C}_{ik}$ is the estimated completion time of o_{cur}.
  if $o_{cur} \neq o_{ik}$ **then**
    $o_{cur} = sm(o_{cur})$; // Move to the next operation in the predecessor chain of o_{ik}.
  end
until $o_{cur} = o_{ik}$;

Finally, Algorithm 3 presents our tabu search method employed in the re-optimization step of our SB approach for JS-TWT. Given a current solution x_k at some iteration of the tabu search, the neighborhood generator $N(x_k)$ in Step 5 may take three different forms. If $N(x_k) = N_{GI}(x_k)$ or $N(x_k) = N_{MAI}(x_k)$, then only the GI or MAI neighborhoods are invoked, respectively. If $N(x_k) = N_{G/MAI}(x_k) = N_{GI}(x_k) \cup N_{MAI}(x_k)$, then both types of neighborhoods are calculated around the current solution x_k. In Section 3, we observe that the combined neighborhood leads to significantly improved performance. For the combined

Algorithm 3: Re-optimization by tabu search in the SB algorithm for JS-TWT.

input : Current disjunctive graph G', the associated operation and job completion times, the tail lengths from every node to all terminal nodes, and the objective function value.

1 x_0 is the initial solution defined by G', $z(x_0)$ is the associated objective value;
2 $x* = x_0$ is the current best solution, $z* = z(x_0)$ is the current best objective value;
3 Initialize the tabu list as $TL = \emptyset$, *terminate* = *false*, set the iteration counter $k = 0$;
4 **while** *do not terminate* **do**
5 Identify all solutions in the neighborhood $N(x_k)$ of x_k. Determine the priority of each $x \in N(x_k)$ depending on the critical path(s) it belongs to;
6 Evaluate at least l_1^{TS} and at most u_1^{TS} solutions in $N(x_k)$ in non-increasing order of their priorities. Pick the next solution x';
7 **if** (x' *is not available*) **or** (z^* *is not improved over the last* u_2^{TS} *iterations*) **then**
8 *terminate* = *true*;
9 **else**
10 Update x^*, z^* if necessary. Update the tabu list TL;
11 $x_{k+1} = x'$, $k = k + 1$;
12 **end**
13 **end**

neighborhood, we exclude solutions in $N_{MAI}(x_k)$ resulting from only the primary interchange performed in Step 1 of Algorithm 1. Recall that if neither $pm(o_{ij})$ nor $sm(o_{ik})$ are part of the critical path, then such a swap cannot decrease the critical path length as discussed earlier in this section. On the other hand, adjacent pairwise interchanges that may potentially lead to reductions in the critical path length are already part of the GI neighborhood.

In contrast to the classical makespan minimization problem in a job shop, a disjunctive arc may participate in several different critical paths in JS-TWT, and each of these critical paths may contribute differently to the overall objective function. Thus, we prioritize the interchanges in the current neighborhood in Step 5 before we evaluate each of these solutions by longest path calculations. To this end, a weight is assigned to each critical path, and the priority of an interchange is determined by the sum of the weights of the critical paths it affects. We experimented with several different weight functions for the critical paths and concluded that the best performing one for a critical path from S to V_j is given by: $U_j = \left\{ \begin{array}{ll} 1, & \text{if } C_j(G') > d_j, \text{ and} \\ 0, & \text{otherwise} \end{array} \right\}$. Alternate measures involve assigning an equal weight to each critical path, regardless of whether the corresponding job is tardy or not, or accounting for the tardiness weight of the corresponding job. In Step 6, we evaluate at least l_1^{TS} and no more than u_1^{TS} solutions in the neighborhood in non-increasing order of their priorities, where u_1^{TS} is set to a fixed percentage of the size of the neighborhood. We stop exploring the neighborhood if a move that improves on $z(x_k)$ is identified. We implement a *first-improve* neighborhood search strategy instead of a *best-improve* strategy that would traverse the entire neighborhood before selecting the next solution due to the computational burden imposed by a potentially large number of neighboring solutions. The best move among those evaluated is picked as the next solution x', provided that it is either not a tabu move or it is a tabu move that exceeds the aspiration criterion which is defined as the best objective value achieved so far. The stopping criterion based on u_2^{TS} in Step 7 is a function of the number of disjunctive arcs fixed in the current disjunctive graph G'. The rationale here is that the size of the search neighborhoods and the complexity of the problem increases steeply as more machines are scheduled during the course of the SB heuristic. The exact forms of l_1^{TS}, u_1^{TS}, and u_2^{TS} will be defined in our numerical study. An important enhancement is left out in the description in Algorithm 3. Once the `while`-loop in Steps 4-13 terminates under the weight function U_j, $j = 1, \ldots, n$, described above, we continue the search by setting $U_j = 0$ for all j for a fixed number of iterations (5 iterations seems to work well in practice) starting from the current best solution x^* before re-invoking the tabu search with the original weight function. This procedure helps to diversify the search around x^*, and if two successive executions of the `while`-loop with the original weight function do not improve z^*, then the re-optimization step is completed. Finally, we explain the details regarding the tabu moves and the tabu list. For a forward interchange performed

on o_{ij_k} and o_{ij_b} in a block of disjunctive arcs $\{(o_{ij_1}, o_{ij_2}), \ldots, (o_{ij_{k-1}}, o_{ij_k}), (o_{ij_k}, o_{ij_{k+1}}), \ldots, (o_{ij_{b-1}}, o_{ij_b})\}$, a backward interchange on $o_{ij_{k+1}}$ and o_{ij_k} is added to the tabu list. Similarly, for a backward interchange performed on o_{ij_1} and o_{ij_k}, a forward interchange on o_{ij_k} and $o_{ij_{k-1}}$ is inserted into the tabu list. For any adjacent pairwise interchange on o_{ij_k} and $o_{ij_{k+1}}$ carried out either in the MAI neighborhood or as part of a forward or a backward interchange on adjacent operations in the GI neighborhood, three reverse moves are included in the tabu list: a forward interchange on $o_{ij_{k+1}}$ and o_{ij_k}, a backward interchange on $o_{ij_{k+1}}$ and o_{ij_k}, and a primary interchange on $o_{ij_{k+1}}$ and o_{ij_k} in the MAI neighborhood. Note that secondary interchanges in the MAI neighborhood are not checked against the tabu list. In our tabu search algorithm, the length of the tabu list is not fixed, but a tabu tenure is associated with each tabu move inserted into the list. That is, a move is deleted from the list after a fixed number of iterations. The tabu tenure is a function of the number of disjunctive arcs in G' at the time the move is added to the tabu list. We conclude this section by pointing out that the number of parameters in our tabu search algorithm is kept minimal which is a significant advantage. Note that one of the major criticisms against metaheuristics is that their performance generally depends on a large number of parameters that need to be tuned properly which is a hard and time consuming endeavor.

2.4 Tree Search The SB procedure as described in this paper in its original form up until here terminates in m iterations since a new bottleneck machine is added to the set of scheduled machines $\mathcal{M}^S$ at each iteration. While this method yields reasonably good results frequently and fairly quickly, many researchers have observed that changing the sequence in which the machines are scheduled leads to greatly improved results in general. For instance, both Adams et al. (1988) and Pinedo and Singer (1999) run a search over the set of possible orders in which the machines may be scheduled in their SB heuristics for $Jm//C_{\max}$ and JS-TWT, respectively. In both cases, a partial enumeration tree is constructed over this search space due to the prohibitively large number of permutations of m machines even for small m. We follow suit with these authors and pick different orders in which the machines are added to $\mathcal{M}^S$. Our computational results in Section 3 indicate that the extra effort is well-spent.

Each node $vx(\mathcal{M}^s)$ of the enumeration tree corresponds to an *ordered* set $\mathcal{M}^S$ and the associated disjunctive graph $G'(N, A_C \cup A_D^S)$ in the SB heuristic, where the order in $\mathcal{M}^S$ prescribes the sequence of scheduling the machines in the SB heuristic. At the root node, $\mathcal{M}^S = \emptyset$ and $A_D^S = \emptyset$. A child node is obtained by appending a machine $i \in (\mathcal{M} \setminus \mathcal{M}^S)$ to $\mathcal{M}^S$ and inserting the necessary disjunctive arcs for machine i into A_D^S as a result of solving the associated single-machine subproblem. Node $vx(\mathcal{M}^s)$ is at level $\mid \mathcal{M}^s \mid$ of the tree, and the relationship between the level and the number of children of $vx(\mathcal{M}^s)$ is expressed by a vector $\beta = (\beta_0, \ldots, \beta_{m-1})$, where β_l represents the maximum number of children for a node at level l of the tree. Thus, for a tree node $vx(\mathcal{M}^s)$ we rank the machines in $\mathcal{M} \setminus \mathcal{M}^S$ in non-increasing order of their respective subproblem objective function values and create a child node for each of the $\beta_{|\mathcal{M}^S|}$ most critical machines. The vector β is generally selected such that the number of children are decreasing with l. This is in alignment with the fundamental idea of the SB heuristic that the earlier scheduling decisions matter more and is also followed by Adams et al. (1988). In contrast, Pinedo and Singer (1999) set $\beta_l, l = 1, \ldots, m-1$, to a constant. The size of the partial enumeration tree is a major determinant of the overall running time of the SB heuristic, and we cannot generally afford more than two or three children per node.

We follow a depth-first-search (DFS) strategy in our partial enumeration. Our primary incentive here is to get a feasible solution for JS-TWT early during the search procedure. The objective value of the current best feasible schedule is then employed in our fathoming rule which further restricts the size of the search tree and reduces the running time. To this end, we define an $m \times m$ matrix F, where $F_{il}, i = 1, \ldots, m,\ l = 1, \ldots, m$, is an estimate of the increase in the objective function if machine i is scheduled in lth order in the SB heuristic. Whenever machine i is appears in lth order in $\mathcal{M}^S$ at a node $vx(\mathcal{M}^s)$, we update F_{il} as

$$F_{il} = \min(F_{il}, \Delta), \tag{16}$$

where Δ represents the difference of the objective values associated with $vx(\mathcal{M}^s)$ and its parent node. Then, a conservative estimate of the lowest objective function value that may be obtained by extending the schedule associated with $vx(\mathcal{M}^s)$ to a feasible schedule for JS-TWT is given by

$$\underline{z}(vx(\mathcal{M}^s)) = z(vx(\mathcal{M}^s)) + \sum_{i \in \mathcal{M} \setminus \mathcal{M}^s} \left(\min_{|\mathcal{M}^s|+1 \le l \le m} F_{il} \right), \tag{17}$$

where $z(vx(\mathcal{M}^s))$ is the objective value associated with $vx(\mathcal{M}^s)$. Given the best available objective value z^* for JS-TWT, we fathom $vx(\mathcal{M}^s)$ if $\underline{z}(vx(\mathcal{M}^s)) \geq z^*$. For the reliability of the estimate in (17), we only apply the fathoming rule if each machine $i \in (\mathcal{M} \setminus \mathcal{M}^S)$ has been scheduled at least l_1^{tree} times at one of the levels $| \mathcal{M}^S | +1, \ldots, m$, before. Moreover, we do not invoke the fathoming rule at levels smaller than a predetermined threshold value l_2^{tree}. The exact values of these parameters of l_1^{tree} and l_2^{tree} are specified in Section 3. To the best of our knowledge, this control structure for fathoming nodes in the partial enumeration tree which incorporates an estimate of the impact of scheduling the currently unscheduled machines is novel for SB algorithms. A short description of how a node is processed in the partial enumeration tree is given in Algorithm 4. The lower bound in Step 10 is computed based on the operation and job completion times in $G'(N, A_C)$ with all machine capacity constraints relaxed during the initialization step of the SB heuristic. Also note that the DFS is implemented as a stack, i.e., according to the last-in-first-out principle.

Algorithm 4: Processing a node in the partial enumeration tree in the SB heuristic.

input : Pick the first unprocessed node in the DFS stack. Assume that the information associated with the parent node $vx(\mathcal{M}^S)$ is available, and machine $i^b \in (\mathcal{M} \setminus \mathcal{M}^S)$ is to be scheduled next.

1 $\mathcal{M}^S = \mathcal{M}^S \cup \{i^b\}$. The operation processing sequence on i^b is retrieved from the corresponding subproblem previously solved while processing the parent node. Update $G'(N, A_C \cup A_D^S)$ accordingly;
2 Determine the longest paths and retrieve the associated operation and job completion times, the tail lengths from every node to all terminal nodes, and the objective function value;
3 Apply re-optimization by tabu search. Update the best known overall objective value z^* and the best schedule x^* if appropriate;
4 Update $F_{i^b|\mathcal{M}^S|}$ per (16);
5 **if** *current node is not to be fathomed per* (17) **then**
6 Set up and solve one single-machine subproblem for each machine $i \in (\mathcal{M} \setminus \mathcal{M}^S)$;
7 Sort the subproblem objectives in non-increasing order of their criticality;
8 Create one child node for the $\min(\beta_{|\mathcal{M}^S|}, | \mathcal{M} \setminus \mathcal{M}^S |)$ most critical machines, and add these in non-decreasing order of criticality to the DFS stack; `/* Ensure that more critical child nodes are processed first in later iterations. */`
9 **end**
10 Terminate the SB heuristic if z^* is equal to the lower bound on the optimal objective value of JS-TWT;

We conclude this section by discussing the synergy between the tree search described above and the tabu search applied during the re-optimization step. At any leaf node of the search tree, the tabu search is applied to a feasible schedule for JS-TWT that results from a distinct order of scheduling the machines. Thus, we may also think of the tree search over the sequence of scheduling the machines as a long-term memory in the context of tabu search. This helps us to further diversify the tabu search, leading to significant gains in solution quality in many cases. In addition, we can directly control the running time of the SB algorithm by controlling the size of the tree which is determined by the vector β and the fathoming parameters. Our entire framework relies on a deterministic mechanism, and thus we can guarantee that the SB algorithm will not terminate with a worse solution if we expand the search tree, barring rare corner cases resulting from fathoming. In our opinion, combined with repeatability this is a significant advantage of our approach in comparison to other (meta-)heuristics based on random search operators.

3. Computational Study We designed our computational experiments with several goals in mind. In the first part of our study, we demonstrate that there exists a synergy between the tabu search in Section 2.3 employed during the re-optimization step in our SB-TS heuristic and the tree search in Section 2.4. In addition, we provide evidence that the combined *generalized interchange/multiple adjacent interchange* (G/MAI) neighborhood in the tabu search performs on average better than both the GI and MAI neighborhoods put into use individually. The GI neighborhood appears to be superior to the MAI neighborhood in general. These results should prompt more research on neighborhood generators for JS-TWT (and for $Jm//C_{\max}$) that reverse several disjunctive arcs in a single move as well as on

integrated neighborhoods with complementary properties. Furthermore, if the subproblem definition is appropriate and the associated solution procedure is effective, then we expect to come across high-quality solutions early during the SB-TS heuristic. Results pointing in this direction are provided. In the second part of our study, we focus on the single-machine subproblems solved in the SB-TS algorithm. We study the quality of the solutions obtained from the subproblems by comparing them to the corresponding optimal solutions. One potential drawback of our subproblem definition stems from the fact that the planning horizon in the transportation problem depends on the sum of the operation processing times. As mentioned in Section 2.2, we argue that this issue may be partially addressed by scaling the down the original due dates, ready times, and the processing times appropriately, and then applying SB-TS to the scaled instance. The resulting job processing sequences of the machines are then fed back into the original instance. We develop this idea in more detail in this section, and our numerical results indicate that this approach does not lead to a major loss in solution quality while reducing computation times. In the final part of our study, we compare our algorithms against existing approaches in the literature based on standard benchmark instances for JS-TWT. SB-TS can also solve classical makespan instances with no modifications required, and we present a limited set of results for well-known hard instances of $Jm//C_{\max}$.

The standard set of benchmark instances for JS-TWT with 10 machines and 10 jobs are due to Pinedo and Singer (1999) who modify 22 well-known instances of $Jm//C_{\max}$ by adding due dates and unit tardiness weights for their purposes. For job j, the due date is set as $d_j = r_j + \lfloor f * \sum_{i=1}^{m} p_{ij} \rfloor$, where f is referred to as due date tightness factor, and assumes one of the values 1.3, 1.5, or 1.6. The unit tardiness weights are set to 1, 2, and 4 for 20%, 60%, and 20% of the jobs, respectively, in line with the distribution of customer order priorities observed in practice. The optimal solutions for these instances are obtained by Singer and Pinedo (1998); however, it appears that in this paper the branch-and-bound algorithm was either stopped prematurely or the due dates were inadvertently set too tight because solutions better than those reported by Singer and Pinedo (1998) appear in subsequent research. Kreipl (2000), De Bontridder (2005), and Essafi et al. (2008) all demonstrate the quality of their algorithms on this set of instances, and we follow suit. In addition, Essafi et al. (2008) create a new set of benchmark instances for JS-TWT based on the instances created by Lawrence (1984) and frequently used for $Jm//C_{\max}$. These instances cover a range of sizes from 5×10 ($m \times n$) to 10×30, and Essafi et al. (2008) adapt these instances to JS-TWT by following the procedure described above. We also present results for this new set of instances.

The algorithms we developed were implemented in Visual Basic (VB) under Excel. The transportation problems were solved by `IBM ILOG CPLEX 9.1` through the VB interface provided by the `IBM ILOG OPL 3.7.1` Component Libraries. The numerical experiments were performed on a single core of an HP Compaq DX 7400 computer with a 2.40 GHz Intel Core 2 Quad Q6600 processor and 3.25 GB of memory running on Windows XP. The Excel/VB environment was selected for ease and speed of development at the expense of computational speed. The integer operation benchmarks Standard Performance Evaluation Corporation (1996) and Standard Performance Evaluation Corporation (2007) published by the Standard Performance Evaluation Corporation indicate that our workstation is about 6.4 times faster than the Dell XPS P90c computer used by Pinedo and Singer (1999) in order to solve their benchmark instances for JS-TWT. On the other hand, based on our comparisons for simple but intensive computing tasks, an equivalent C program is about 6.75 times faster than the corresponding Excel/VB code. That is, we reckon that our implementation environment is approximately as slow as that of Pinedo and Singer (1999). This point should be taken into account while evaluating the times reported in our study.

A typical and well-deserved criticism against approaches based on metaheuristics argues that the number of parameters used in such algorithms is too many, and the performance depends crucially on an extensive parameter tuning. Thus, as a design goal we refrain from parametrizing too much and set the values of most of our parameters once and for all at the beginning for all instances. In Step 6 of Algorithm 3, we explore at least $l_1^{TS} = 30$ solutions in the neighborhood of the current solution x_k, unless this number exceeds the size of the neighborhood, and do not evaluate more than 60% of the total number solutions in the neighborhood as long as this number u_1^{TS} is larger than 30. In Step 7, we terminate the tabu search if the best known solution does not improve for u_2^{TS} successive iterations, where u_2^{TS} is determined as 0.3 times the number of disjunctive arcs present in the current disjunctive graph G'. Similarly, the tabu tenure of a reverse move added to the tabu list is calculated as 0.75 times the number of disjunctive arcs in G'. The values of these parameters were determined by some preliminary

runs during the development process, but no experimental design was applied. The dependence of u_2^{TS} and the tabu tenure on the number of disjunctive arcs in G' reflects the extra effort required to escape local optima as more machines are scheduled. The parameters controlling the size of the search tree are determined by the number of machines in an instance and are given separately for each set of instances in the sequel.

3.1 Tree Search, Tabu Search, and Neighborhood Generators In our first set of experiments, we solve 22 instances of size 10×10 due to Pinedo and Singer (1999) using three types of neighborhoods for different tree sizes. The results are reported in Table 1. The table consists of three parts, one for each possible value of $f = 1.3, 1.5$, and 1.6. The instance names are listed in the first column and the associated optimal objective values from Singer and Pinedo (1998) are given in the next column. The associated value of f is appended to the name of the instance. As discussed in Section 2.4, the size of the search tree for identifying a good sequence of scheduling the machines in the SB heuristic is controlled by a parameter $\beta = (\beta_0, \ldots, \beta_{m-1})$, where β_l represents the maximum number of children for a node at level l of the tree. In our experiments, β assumes one of five possible values $(1,1,1,1,1,1,1,1,1,1)$, $(2,2,2,1,1,1,1,1,1,1)$, $(3,2,2,2,2,2,1,1,1,1)$, $(3,2,2,3,2,2,1,1,1,1)$, and $(3,2,2,3,3,2,1,1,1,1)$. Moreover, the values of the parameters l_1^{tree} and l_2^{tree} which control the fathoming rule in (17) are set to 3 and 5, respectively. For each value of β, the SB-TS algorithm is run three times on a given instance by selecting the neighborhood generator in the tabu search as GI, MAI, or the combined neighborhood G/MAI. The objective function values for each combination of the possible values of β and the neighborhood generator are presented in columns 6-15 in Table 1. For these experiments, re-optimization is invoked each time a new machine is scheduled. This is referred to as full re-optimization and indicated by "RF" attached to β in the column headers of Table 1. In addition, in order to set the baseline and illustrate the synergy between the tabu search and the tree search we first obtain a feasible solution for JS-TWT by selecting only one bottleneck machine at each level and then apply tabu search to this feasible solution. These results are reported in columns 3-5 under "(1,1,1,1,1,1,1,1,1,1)-RL," where "RL" stands for "re-optimization at a leaf node only." In all tables in this section, we designate an optimal solution obtained by a heuristic by appending a "*" to the associated objective value. Furthermore, objective values which are strictly better than those reported by Singer and Pinedo (1998) appear in bold. The average optimality gap over 22 associated instances and the number of instances solved to optimality are presented in rows with headers "Avg. Gap(%)" and "# Opt.," respectively. A row with the header "Avg. T.(sec.)" provides the average time elapsed until the best solution is identified in a given run in real seconds over 22 associated instances. Note that a positive objective value obtained by SB-TS for an instance with zero optimal objective value has to be excluded from the computation of the average optimality gap. Thus, as an alternate measure we calculate the sum of the weighted tardiness values for 22 instances and compute the gap of this number with respect to the corresponding sum of the optimal objective values. This performance measure is labeled as "Total Gap(%)."

The results reveal several important observations and trends. First, the combined neighborhood G/MAI is superior to both the GI and MAI neighborhoods for all instances and across all performance measures when these neighborhoods are used in isolation. These results verify our claims in Section 2.3 that the GI and MAI neighborhoods feature complementary properties and combining them incorporates a degree of diversification directly into the neighborhood definition. Furthermore, the GI neighborhood consistently outperforms the MAI neighborhood for $f = 1.3$. For instances with looser due dates, the picture is more mixed. Second, we observe that the sequence of scheduling the machines in the shifting bottleneck heuristic has a significant effect on the solution quality. Trying out different orders is crucial to the overall success of the SB-TS algorithm. The trade-off between solution quality and solution time is clearly illustrated as we move from (1,1,1,1,1,1,1,1,1,1)-RL toward (3,2,2,3,3,2,1,1,1,1)-RF in Table 1. Note that the tree parameters are selected such that the search tree under the setting (1,1,1,1,1,1,1,1,1,1)-RF is a subset of the search tree under the setting (2,2,2,1,1,1,1,1,1,1)-RF, and so on, and the solution quality improves from left to right in Table 1. However, the benefits of using a larger tree level off from (3,2,2,2,2,2,1,1,1,1)-RF to (3,2,2,3,2,2,1,1,1,1)-RF and then to (3,2,2,3,3,2,1,1,1,1)-RF. That is, the tree parameters provide us with a powerful tool to trade-off solution quality and solution time given the deterministic nature of SB-TS, and the tree search may be regarded as a long-term memory from the perspective of tabu search. The results under (1,1,1,1,1,1,1,1,1,1)-RL clearly state that generating an initial feasible solution for JS-TWT and then applying the proposed tabu search yields poor solutions.

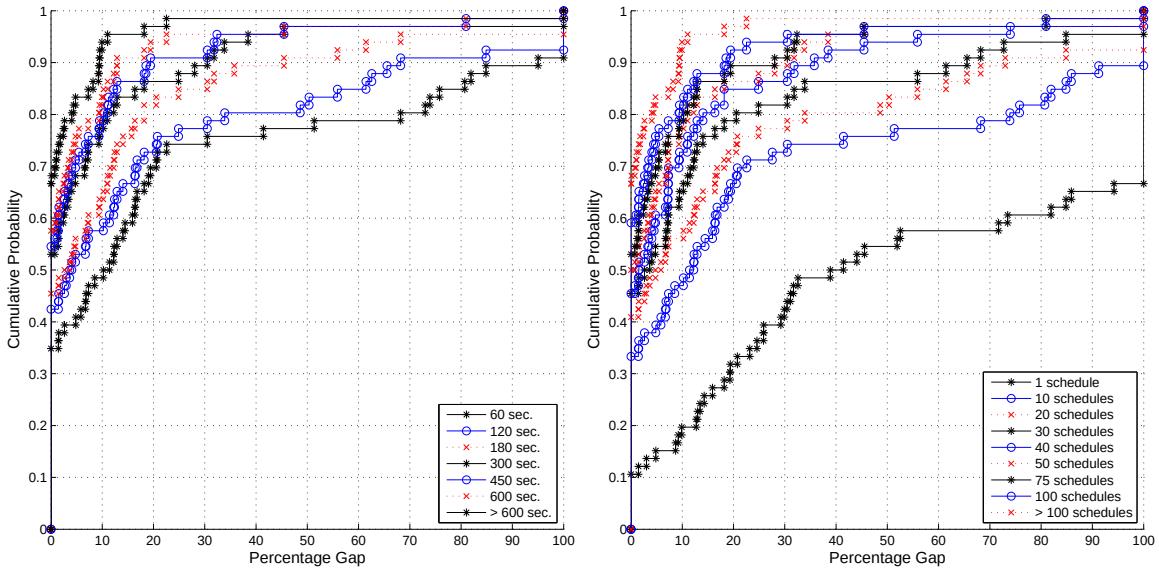
Table 1: Effect of the tree parameters and the neighborhood generators on the performance of the SB-TS heuristic.

		(1,1,1,1,1,1,1,1,1,1)-RL			(1,1,1,1,1,1,1,1,1,1)-RF			(2,2,2,1,1,1,1,1,1,1)-RF			(3,2,2,2,2,2,1,1,1,1)-RF			(3,2,2,3,2,2,1,1,1,1)-RF			(3,2,2,3,3,2,1,1,1,1)-RF		
Instance	**Opt.**	**MAI**	**GI**	**G/MAI**	**MAI**	**GI**	**G/MAI**	**MAI**	**GI**	**G/MAI**	**MAI**	**GI**	**G/MAI**	**MAI**	**GI**	**G/MAI**	**MAI**	**GI**	**G/MAI**
abz05_1.3	1405	2554	2252	2467	2082	1957	2024	1569	1462	1608	1563	1462	1487	1445	1462	1462	1445	1462	1462
abz06_1.3	436	937	1219	930	1071	678	578	516	521	449	447	449	436*	447	449	436*	436*	436*	436*
la16_1.3	1170	1430	1783	1367	1570	1519	1337	1250	**1169***	1250	**1169***	**1169***	**1169***	**1169***	**1169***	**1169***	**1169***	**1169***	**1169***
la17_1.3	900	1017	1227	1219	1073	1353	1017	1073	936	**899***	**899***	**899***	**899***	**899***	**899***	**899***	**899***	**899***	**899***
la18_1.3	929	1518	1177	1061	1521	963	974	936	963	963	929*	929*	929*	929*	929*	929*	929*	929*	929*
la19_1.3	948	1467	1723	1441	2083	1226	1440	1332	1226	1129	1146	1036	955	1146	1036	955	1111	1024	955
la20_1.3	809	1117	1440	922	987	1136	878	863	929	**805***	**805***	**805***	**805***	**805***	**805***	**805***	**805***	**805***	**805***
la21_1.3	464	898	1283	1140	604	912	**463***	524	**463***	**463***	**463***	**463***	**463***	**463***	**463***	**463***	**463***	**463***	**463***
la22_1.3	1068	1838	2207	1796	1084	1268	1344	1084	1084	1084	1084	1084	1084	1084	1084	1084	1084	1084	1084
la23_1.3	837	1581	1997	1269	1220	1095	970	1111	1012	877	971	877	877	970	877	877	929	877	877
la24_1.3	835	1163	1086	857	979	1091	1051	855	946	839	855	835*	835*	839	835*	835*	835*	835*	835*
mt10_1.3	1368	1806	1710	1786	1767	1814	1777	1767	1685	1638	1653	**1363***	**1363***	1397	**1363***	**1363***	1397	**1363***	**1363***
orb01_1.3	2568	4466	3608	3608	3252	2933	2914	3252	2605	2914	2839	2568*	2630	2839	2568*	2630	2763	2568*	2630
orb02_1.3	1412	1856	2077	1990	2418	1774	1842	1663	1671	1659	**1408***	**1408***	**1408***	**1408***	**1408***	**1408***	**1408***	**1408***	**1408***
orb03_1.3	2113	4039	4088	3406	3371	3408	3223	2846	3079	2719	2222	2176	2186	2222	2176	2115	2221	2176	2115
orb04_1.3	1623	2750	2316	2502	2482	1977	1999	1676	1963	1999	1676	1844	1652	1676	1844	1652	1676	1806	1652
orb05_1.3	1593	2290	2388	2174	1840	2044	2098	1840	1593*	1761	1738	1593*	1667	1593*	1593*	1593*	1593*	1593*	1593*
orb06_1.3	1792	4008	3651	3527	3371	3334	3077	2253	2059	2090	1982	1844	**1790***	1982	1843	**1790***	1982	1794	**1790***
orb07_1.3	590	1003	1003	1003	838	798	735	626	672	626	616	658	616	616	657	616	616	642	616
orb08_1.3	2429	4348	4841	4307	3893	4250	2898	2541	3024	2580	2541	2624	2503	2503	2661	2503	2503	2661	2453
orb09_1.3	1316	2564	2453	2047	1661	1639	1571	1661	1592	1571	1316*	1576	1316*	1316*	1316*	1316*	1316*	1316*	1316*
orb10_1.3	1679	2353	1959	2079	2230	2810	2444	1877	2157	1895	1813	1937	1801	1809	1801	1801	1809	1759	1801
Total Gap(%)		66.18	67.90	51.67	46.36	41.35	29.59	17.08	16.01	12.49	6.54	4.65	2.08	4.50	3.37	1.47	3.91	2.78	1.30
Avg. Gap(%)		65.06	74.62	53.49	47.81	40.78	27.20	16.14	14.69	10.33	5.90	4.33	1.74	4.08	3.12	1.29	3.42	2.47	1.20
# Opt.		0	0	0	0	0	1	0	3	3	7	10	11	8	11	12	10	12	12
Avg. T.(sec.)		3.93	3.87	4.34	6.26	6.91	8.00	25.15	26.74	31.53	139.88	167.00	180.39	223.60	191.29	282.63	304.84	267.63	356.19
abz05_1.5	69	225	221	186	166	110	75	114	96	75	75	75	70	75	75	70	69*	69*	69*
abz06_1.5	0	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
la16_1.5	166	284	336	283	322	270	239	168	169	178	166*	168	166*	166*	168	166*	166*	166*	166*
la17_1.5	260	304	397	289	260*	268	268	260*	260*	260*	260*	260*	260*	260*	260*	260*	260*	260*	260*
la18_1.5	34	150	351	200	83	74	34*	36	60	34*	34*	34*	34*	34*	34*	34*	34*	34*	34*
la19_1.5	21	297	434	216	56	110	208	56	54	104	42	52	23	42	42	23	42	21*	23
la20_1.5	0	87	143	57	1	14	1	1	1	1	1	1	1	1	1	1	1	1	1
la21_1.5	0	86	52	78	12	14	12	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
la22_1.5	196	509	492	504	604	245	340	196*	196*	196*	196*	196*	196*	196*	196*	196*	196*	196*	196*
la23_1.5	2	55	54	2*	16	20	2*	2*	2*	2*	2*	2*	2*	2*	2*	2*	2*	2*	2*
la24_1.5	82	328	234	162	138	115	131	107	94	106	82*	82*	82*	82*	82*	82*	82*	82*	82*
mt10_1.5	394	582	784	742	794	638	433	495	427	433	444	427	394*	444	427	394*	394*	427	394*
orb01_1.5	1098	2158	2034	1835	1970	1700	1326	1819	1700	1326	1442	1238	1326	1381	1238	1202	1330	1238	1202

		(1,1,1,1,1,1,1,1,1,1)-RL			(1,1,1,1,1,1,1,1,1,1)-RF			(2,2,2,1,1,1,1,1,1,1)-RF			(3,2,2,2,2,2,1,1,1,1)-RF			(3,2,2,3,2,2,1,1,1,1)-RF			(3,2,2,3,3,2,1,1,1,1)-RF		
Instance	**Opt.**	**MAI**	**GI**	**G/MAI**	**MAI**	**GI**	**G/MAI**	**MAI**	**GI**	**G/MAI**	**MAI**	**GI**	**G/MAI**	**MAI**	**GI**	**G/MAI**	**MAI**	**GI**	**G/MAI**
orb02_1.5	292	920	1211	690	442	422	413	330	345	322	292*	292*	322	292*	292*	322	292*	292*	292*
orb03_1.5	918	2981	2548	2578	1872	1571	1275	1021	1220	1098	991	928	968	991	928	952	991	928	928
orb04_1.5	358	924	1223	1011	358*	631	977	358*	448	358*	358*	358*	358*	358*	358*	358*	358*	358*	358*
orb05_1.5	405	1242	1154	1059	600	557	531	529	547	405*	405*	441	405*	405*	441	405*	405*	441	405*
orb06_1.5	426	1187	1014	769	1180	1034	775	536	569	775	524	520	426*	524	426*	426*	524	426*	426*
orb07_1.5	50	238	270	264	134	205	141	132	144	50*	50*	50*	50*	50*	50*	50*	50*	50*	50*
orb08_1.5	1023	2206	1751	1519	1906	1767	1023*	1539	1414	1023*	1292	1399	1023*	1205	1340	1023*	1205	1340	1023*
orb09_1.5	297	1062	984	1086	525	394	306	306	374	297*	297*	334	297*	297*	330	297*	297*	330	297*
orb10_1.5	346	1092	1300	1131	747	918	672	575	553	615	537	424	424	458	424	424	458	424	424
Total Gap(%)		162.81	163.90	127.76	89.31	72.08	42.64	33.29	34.74	18.97	16.36	13.11	6.06	12.83	10.52	3.88	11.17	10.07	3.03
Avg. Gap(%)		358.58	423.50	189.44	126.51	135.07	87.78	34.53	38.53	31.46	12.63	13.45	3.33	10.87	9.79	2.71	9.63	4.56	2.03
# Opt.		1	1	2	3	1	4	6	5	11	13	10	15	13	11	15	15	14	17
Avg. T.(sec.)		8.99	9.45	12.04	10.69	12.45	11.13	41.25	45.13	30.11	134.54	195.79	106.31	186.22	231.90	155.68	260.79	319.30	260.67
abz05_1.6	0	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
abz06_1.6	0	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
la16_1.6	0	0*	48	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
la17_1.6	65	65*	65*	65*	65*	65*	65*	65*	65*	65*	65*	65*	65*	65*	65*	65*	65*	65*	65*
la18_1.6	0	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
la19_1.6	0	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
la20_1.6	0	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
la21_1.6	0	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
la22_1.6	0	87	82	4	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
la23_1.6	0	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
la24_1.6	0	50	16	0*	21	18	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
mt10_1.6	141	184	184	184	184	626	287	184	186	184	176	184	155	176	184	155	176	184	155
orb01_1.6	566	1524	1334	862	1169	875	1113	1046	875	881	836	829	776	836	619	776	821	619	619
orb02_1.6	44	346	354	234	86	132	94	86	104	58	58	58	52	58	58	52	58	58	52
orb03_1.6	422	1164	1311	782	779	645	1013	602	605	546	533	533	461	533	533	461	533	461	461
orb04_1.6	66	143	259	143	96	96	66*	90	66*	66*	66*	66*	66*	66*	66*	66*	66*	66*	66*
orb05_1.6	163	295	275	279	330	458	303	181	211	193	181	193	181	181	193	181	181	193	181
orb06_1.6	31	775	1367	652	304	388	275	84	69	67	38	38	31*	38	38	31*	37	38	31*
orb07_1.6	0	34	12	0*	6	16	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*	0*
orb08_1.6	621	1312	1702	1282	1025	1072	1325	883	795	843	716	716	701	675	716	672	675	716	672
orb09_1.6	66	239	290	233	157	181	111	111	166	111	66*	66*	66*	66*	66*	66*	66*	66*	66*
orb10_1.6	76	220	169	187	578	396	88	96	198	88	84	124	84	84	108	84	84	92	78
Total Gap(%)		184.74	230.30	117.03	112.30	119.73	109.64	51.61	47.72	37.20	24.68	27.02	16.67	22.87	17.03	15.39	22.16	13.14	8.18
Avg. Gap(%)		222.23	354.89	156.98	110.46	134.87	69.28	27.66	34.47	18.25	8.64	11.57	4.95	8.34	8.93	4.74	8.07	7.19	3.12
# Opt.		9	8	11	10	10	13	12	13	13	14	14	15	14	14	15	14	14	15
Avg. T.(sec.)		11.51	8.50	8.92	12.02	13.72	12.96	24.17	30.37	26.32	110.04	165.57	149.35	143.81	219.97	229.28	199.91	317.74	223.46

The bottom line is that a synergy is created by embedding the tabu search into the shifting bottleneck framework. Third, SB-TS yields excellent solutions in reasonable times. Over 66 instances, the SB-TS heuristic with the tree parameters (3,2,2,3,2,2,1,1,1,1)-RF and the combined neighborhood generator G/MAI provides 42 optimal solutions with an average optimality gap of 2.92% in an average computation time of 223 seconds. The corresponding performance measures for (3,2,2,3,3,2,1,1,1,1)-RF and G/MAI are 44 optimal solutions, an average optimality gap of 2.12%, and an average computation time of 280 seconds, respectively. In Figure 8, we present a detailed analysis of the solution quality versus the solution time and the number of feasible schedules constructed for JS-TWT. For the results with the tree parameters (3,2,2,3,3,2,1,1,1,1)-RF and the G/MAI neighborhood, we take a snapshot of the optimality gaps after 60, 120, 180, 300, 450, and 600 seconds of solution time and depict the empirical cumulative distributions of these gaps in Figure 8(a). A similar figure is produced for the number of leaf nodes traversed in the search tree in Figure 8(b), where each leaf node corresponds to a feasible schedule for JS-TWT. In these figures, gaps larger than 100% appear as 100%. Also, if the objective value is positive for an instance with a zero optimal objective value, the gap is set to 100%. In Figure 8(a), we observe that more than 1/3 of the instances (23 instances) are solved to optimality in 60 seconds, while this figure increases to 42% (28 instances) in 120 seconds. After 600 seconds of solution time, 58% of the instances (38 instances) are solved to optimality, and 76% (50 instances) and 83% (55 instances) of the instances are within 5% and 10% of optimality, respectively. Intuitively, if the shifting bottleneck framework performs well, then good solutions should be identified early in the tree. This intuition is verified in Figure 8(b). For 11% of the instances (7 instances), an optimal solution is obtained after re-optimization at the first leaf node, while 1/3 of the instances (22 instances) are solved to optimality after visiting at most 10 leaf nodes in the search tree.

One specific idea that involves scaling the data is discussed in the next section in order to reduce the solution times. In addition, there are several avenues we may explore in the future in order to decrease the computational effort expended in SB-TS. Computing the longest paths approximately in order to evaluate the solutions in the neighborhood of the current solution in the tabu search would be a first priority because a significant portion of the total solution time in SB-TS is devoted to computing longest paths from scratch. See Balas and Vazacopoulos (1998) and Essafi et al. (2008) for approximate move evaluations. We also reckon that SB-TS is amenable to parallelization as different branches in the search tree may be assigned to different processing units. Finally, it may be possible to do warm starts in the subproblems if similar instances of the transportation problem are solved at different points in time during the course of the shifting bottleneck algorithm.

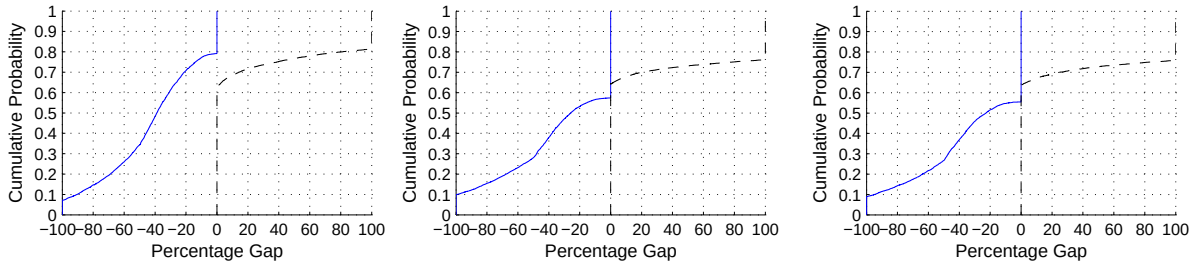


(a) Optimality gaps as a function of solution times.

(b) Optimality gaps as a function of the number of feasible schedules constructed for JS-TWT.

Figure 8: The progress of the optimality gaps in the SB-TS heuristic.

3.2 Subproblems In this section, we first analyze the quality of the preemptive lower bounds and the feasible solutions obtained for the subproblems in the SB-TS heuristic. To this end, we export all 22,281 subproblems solved during the course of the shifting bottleneck algorithm while applying SB-TS to the instances in Table 1 with the tree parameters (2,2,2,1,1,1,1,1,1,1)-RF and the G/MAI neighborhood. We solve these instances optimally using a standard single-machine time-indexed formulation, first introduced by Dyer and Wolsey (1990), and plot the empirical cumulative distribution of the optimality gaps in Figure 9. The nonpositive gaps are associated with the lower bounds given by the transportation problem, and the nonnegative gaps correspond to the the feasible non-preemptive solutions constructed based on the processing sequences extracted from the optimal solution of the transportation problem as discussed in Section 2.2. All gaps larger than 100% appear as 100%, and if the objective function value of a non-preemptive feasible solution is positive for an instance with a zero optimal objective value, the gap is set to 100%. Figure 1 reveals that more than 60% of the subproblems are solved optimally for all values of f while in about 20% of the instances either the objective value is at least twice the optimal objective value or an optimal solution cannot be identified for an instance with a zero optimal objective value. Furthermore, the lower bound obtained from the transportation problem is identical to the optimal non-preemptive objective value in more than 40% of the instances for $f = 1.5, 1.6$, while this number drops to 20% for $f = 1.3$. In about 10% of the instances, the preemptive lower bound is zero while the optimal objective value is positive which corresponds to a -100% optimality gap for the transportation problem. Summarizing, the quality of the information provided by the transportation problem is frequently sufficient to construct good feasible solutions for the single-machine subproblems; however there is room for improvement. For example, instances with zero lower bounds are difficult. In such cases, there typically exists a large number of alternate optimal solutions in the transportation problem, and extracting a good sequence for constructing a feasible solution is tricky. We note that the lower bound is zero in 67% of the cases in which the gap of the feasible solution is 100% in Figure 9.



(a) $f = 1.3$ (b) $f = 1.5$ (c) $f = 1.6$

Figure 9: Solution quality of the lower and upper bounds obtained from the subproblems.

Table 2: Effect of scaling the processing times on the solution quality.

	$f = 1.3$				$f = 1.5$				$f = 1.6$			
Instance	$\sigma = 1$	$\sigma = 2$	$\sigma = 3$	$\sigma = 4$	$\sigma = 1$	$\sigma = 2$	$\sigma = 3$	$\sigma = 4$	$\sigma = 1$	$\sigma = 2$	$\sigma = 3$	$\sigma = 4$
abz05	1462	1409	1450	1487	69*	69*	72	69*	0*	0*	0*	0*
abz06	436*	449	436*	496	0*	0*	0*	0*	0*	0*	0*	0*
la16	**1169***	**1169***	**1169***	**1169***	166*	180	170	180	0*	0*	0*	0*
la17	**899***	**899***	**899***	939	260*	263	263	260*	65*	84	90	81
la18	929*	936	936	986	34*	34*	34*	66	0*	0*	0*	0*
la19	955	955	1024	948*	23	21*	58	53	0*	0*	0*	0*
la20	**805***	**805***	867	866	1	2	3	13	0*	0*	0*	0*
la21	**463***	**463***	468	475	0*	0*	39	41	0*	0*	0*	0*
la22	1084	1086	1086	1090	196*	196*	196*	281	0*	0*	0*	2
la23	877	**835***	892	879	2*	4	6	2*	0*	0*	0*	0*
la24	835*	835*	835*	835*	82*	94	107	114	0*	0*	0*	0*
mt10	**1363***	**1363***	1397	1397	394*	443	396	413	155	176	177	177
orb01	2630	2568*	2568*	2568*	1202	1282	1141	1309	619	569	667	661
orb02	**1408***	1434	1434	1434	292*	292*	353	378	52	52	52	64
orb03	2115	2194	2314	2200	928	952	918*	918*	461	461	422*	477
orb04	1652	1652	1652	1623*	358*	369	397	397	66*	66*	80	96
orb05	1593*	1593*	1593*	1667	405*	405*	405*	405*	181	181	181	217

	$f = 1.3$				$f = 1.5$				$f = 1.6$			
Instance	$\sigma = 1$	$\sigma = 2$	$\sigma = 3$	$\sigma = 4$	$\sigma = 1$	$\sigma = 2$	$\sigma = 3$	$\sigma = 4$	$\sigma = 1$	$\sigma = 2$	$\sigma = 3$	$\sigma = 4$
orb06	**1790***	1844	2047	1844	426*	426*	426*	524	31*	40	36	36
orb07	616	593	593	665	50*	52	133	133	0*	0*	0*	0*
orb08	2453	2513	2533	2592	1023*	1023*	1075	1062	672	691	698	657
orb09	1316*	1483	1326	1393	297*	297*	297*	297*	66*	66*	66*	105
orb10	1801	1753	1801	1842	424	519	466	464	78	102	186	171
Total Gap(%)	1.30	1.93	3.66	3.93	3.03	7.55	8.05	14.63	8.18	10.04	17.43	21.36
Avg. Gap(%)	1.20	1.64	3.19	4.40	2.03	10.20	32.83	31.43	3.12	7.62	13.89	19.52
# Opt.	12	9	6	5	17	11	7	7	15	13	13	10
Avg. T.(sec.)	356.19	354.79	229.11	253.61	260.67	409.62	204.27	268.41	223.46	203.72	252.88	190.54
Avg. SubT.(sec.)	184.10	87.42	56.95	43.04	255.70	126.74	79.91	53.94	142.79	76.14	51.83	38.98

An important issue regarding the transportation problem is the length of the processing times. As discussed in Section 2.2, the length of the planning horizon in TR depends on the sum of the operation processing times. In other words, for an instance of JS-TWT with long processing times it may be computationally very expensive to solve the subproblems in the SB-TS heuristic. We propose a practical remedy to this problem that retains the solution quality. For a given instance with long processing times, we first create an auxiliary instance by scaling the ready times, processing times, and the due dates as specified below:

$$\begin{aligned} r'_j &= \lceil r_j/\sigma \rceil, \quad \forall j, \\ p'_{ij} &= \lceil p_{ij}/\sigma \rceil, \quad \forall i, j, \\ d'_j &= \lfloor d_j/\sigma \rfloor, \quad \forall j, \end{aligned} \tag{18}$$

where σ is the scaling constant and the data in the scaled instance are denoted by a prime in the superscript. Then, we solve the scaled instance by SB-TS, record the operation processing sequences in the final solution, and then compute the longest paths and the corresponding objective value in the original instance based on these sequences. Observe that the scaling in (18) ensures that the objective value obtained for the original instance is no larger than σ times that of the scaled instance. This property is based on the fact that the length of any path in the scaled instance is no smaller than $1/\sigma$ times that in the original instance, and the scaled due date is no larger than $1/\sigma$ times the original due date. In Table 2, we report the results obtained by applying SB-TS to the instances of Pinedo and Singer (1999) by setting $\sigma = 1, 2, 3, 4$. The tree parameters are selected as (3,2,2,3,3,2,1,1,1,1)-RF, and the combined neighborhood G/MAI is employed. Thus, the results for $\sigma = 1$ are identical to those in the last column of Table 1. The times expended in solving the transportation problems are listed in the last row of Table 2 labeled as "Avg. SubT.", where the reduction is approximately linear in σ. Of course, this reduction does not necessarily translate into a reduction in the solution times until the best solution is identified as demonstrated by the row "Avg. T." The results in Table 2 generally provide evidence that scaling the processing times should be the first choice for shorter solution times in the SB-TS heuristic without compromising the solution quality significantly. Furthermore, scaling sometimes leads to higher quality solutions. Taking the best objective value over σ for each instance, the average optimality gap for $f = 1.3$ is just 0.27% and the number of optimal solutions increases from 12 to 16. The corresponding numbers for $f = 1.5$ and 1.6 are 1.26%, 19 optimal solutions and 2.19%, 16 optimal solutions, respectively. In total, 51 out of 66 instances are solved to optimality as compared to 44 instances in the last column of Table 1.

3.3 Benchmarking Against Existing Algorithms In this section, we present a comparison of the performance of SB-TS against the current state-of-the-art for JS-TWT. We start by benchmarking our algorithm against those by Pinedo and Singer (1999), Kreipl (2000), De Bontridder (2005), and Essafi et al. (2008) on the standard test suite of Pinedo and Singer (1999). Then, we move on to a new set of benchmark instances for JS-TWT introduced by Essafi et al. (2008). Finally, we demonstrate that SB-TS does also perform well on well-known instances of $Jm//C_{\max}$ although it is not tailored to the makespan criterion.

The results for the instances of Pinedo and Singer (1999) are reported in Table 3. For SB-TS, we pick three versions of the algorithm from Table 1. The third, forth, and fifth columns in Table 3 correspond to the results with the combined neighborhood G/MAI and the tree parameters (3,2,2,2,2,2,1,1,1,1)-RF, (3,2,2,3,2,2,1,1,1,1)-RF, and (3,2,2,3,3,2,1,1,1,1)-RF in Table 1, respectively. The results of Pinedo and Singer (1999) appear in the column "SB(10,3)" which is the computationally more intensive version of their

shifting bottleneck algorithm that yields their best results. In a similar way, the column "LSRW(200)" corresponds to the the large step random walk of Kreipl (2000) with longer run times. The output of the genetic local search algorithm by Essafi et al. (2008) is given in the column "GLS." De Bontridder (2005) only report results for $f = 1.5$, and this author's results with the criterion "rbf" are listed in the column "BONT." This criterion produces the best average results obtained by the author. As mentioned in Section 1, the algorithms of Kreipl (2000), De Bontridder (2005), and Essafi et al. (2008) incorporate randomness, and these authors report both best and average results over either 5 or 10 independent runs for each instance. Since SB-TS is deterministic, it is only fair to compare it against the average results, and the results under "LSRW(200)," "GLS," and "BONT" are all average results reported in the original papers. Furthermore, recall that the best result over σ for a given instance in Table 2 leads to significant improvements in some cases. That is, it would be possible to improve the "best" performance of SB-TS by adopting such strategies or adding randomness to our neighborhood traversal rule in the tabu search, etc.

Table 3: Benchmarking against existing approaches on the instances of Pinedo and Singer (1999).

Instance	Opt.	SB-TS†	SB-TS‡	SB-TS$^{\diamond}$	SB(10,3)	LSRW(200)	GLS	BONT
abz05_1.3	1405	1487	1462	1462	1464	1451	**1403***	
abz06_1.3	436	436*	436*	436*	436*	436*	436*	
la16_1.3	1170	**1169***	**1169***	**1169***	1170*	1170*	1175	
la17_1.3	900	**899***	**899***	**899***	900*	900*	900*	
la18_1.3	929	929*	929*	929*	936	929*	933	
la19_1.3	948	955	955	955	955	951	949	
la20_1.3	809	**805***	**805***	**805***	878	809*	**805***	
la21_1.3	464	**463***	**463***	**463***	464*	464*	464*	
la22_1.3	1068	1084	1084	1084	1086	1086	1087	
la23_1.3	837	877	877	877	875	875	865	
la24_1.3	835	835*	835*	835*	835*	835*	835*	
mt10_1.3	1368	**1363***	**1363***	**1363***	1368*	1368*	1372	
orb01_1.3	2568	2630	2630	2630	2890	2616	2651	
orb02_1.3	1412	**1408***	**1408***	**1408***	1412*	1434	1444	
orb03_1.3	2113	2186	2115	2115	2113*	2204	2170	
orb04_1.3	1623	1652	1652	1652	1623*	1674	1643	
orb05_1.3	1593	1667	1593*	1593*	1667	1662	1659	
orb06_1.3	1792	**1790***	**1790***	**1790***	1792*	1802	1792*	
orb07_1.3	590	616	616	616	590*	618	592	
orb08_1.3	2429	2503	2503	2453	2617	2554	2522	
orb09_1.3	1316	1316*	1316*	1316*	1483	1334	1316*	
orb10_1.3	1679	1801	1801	1801	1827	1775	1718	
Total Gap(%)		2.08	1.47	1.30	3.88	2.34	1.58	
Avg. Gap(%)		1.74	1.29	1.20	3.04	1.93	1.17	
# Opt.		11	12	12	11			
# Best Heur.		12	13	14	5	4	8	
abz05_1.5	69	70	70	69*	77	70	69*	78.6
abz06_1.5	0	0*	0*	0*	0*	0*	0*	0*
la16_1.5	166	166*	166*	166*	175	166*	166*	181.3
la17_1.5	260	260*	260*	260*	260*	260*	260*	260.4
la18_1.5	34	34*	34*	34*	34*	34*	34*	34.4
la19_1.5	21	23	23	23	21*	21*	21*	43.8
la20_1.5	0	1	1	1	0*	0*	0*	0.8
la21_1.5	0	0*	0*	0*	0*	0*	0*	4.8
la22_1.5	196	196*	196*	196*	196*	196*	196*	204.1
la23_1.5	2	2*	2*	2*	2*	2*	2*	2*
la24_1.5	82	82*	82*	82*	82*	90	86	89.9
mt10_1.5	394	394*	394*	394*	394*	414	394*	414.7
orb01_1.5	1098	1326	1202	1202	1196	1143	1159	1308.1
orb02_1.5	292	322	322	292*	292*	292*	292*	302.8
orb03_1.5	918	968	952	928	967	965	943	1027.9
orb04_1.5	358	358*	358*	358*	358*	358*	394	446.6
orb05_1.5	405	405*	405*	405*	517	455	405*	491.5
orb06_1.5	426	426*	426*	426*	426*	426*	440	528.7
orb07_1.5	50	50*	50*	50*	50*	119	55	63
orb08_1.5	1023	1023*	1023*	1023*	1023*	1138	1059	1158.1
orb09_1.5	297	297*	297*	297*	331	297*	311	344.5

Instance	Opt.	SB-TS†	SB-TS‡	SB-TS◇	SB(10,3)	LSRW(200)	GLS	BONT
orb10_1.5	346	424	424	424	458	408	400	533.1
Total Gap(%)		6.06	3.88	3.03	6.56	6.48	3.87	16.81
Avg. Gap(%)		3.33	2.71	2.03	4.67	9.32	2.74	18.32
# Opt.		15	15	17	15			
# Best Heur.		15	15	18	15	14	14	2
abz05_1.6	0	0*	0*	0*	0*	0*	0*	
abz06_1.6	0	0*	0*	0*	0*	0*	0*	
la16_1.6	0	0*	0*	0*	0*	0*	0*	
la17_1.6	65	65*	65*	65*	65*	65*	65*	
la18_1.6	0	0*	0*	0*	0*	0*	0*	
la19_1.6	0	0*	0*	0*	0*	0*	0*	
la20_1.6	0	0*	0*	0*	0*	0*	0*	
la21_1.6	0	0*	0*	0*	0*	0*	0*	
la22_1.6	0	0*	0*	0*	0*	0*	0*	
la23_1.6	0	0*	0*	0*	0*	0*	0*	
la24_1.6	0	0*	0*	0*	0*	0*	0*	
mt10_1.6	141	155	155	155	184	144	162	
orb01_1.6	566	776	776	619	619	624	688	
orb02_1.6	44	52	52	52	64	44*	44*	
orb03_1.6	422	461	461	461	461	441	514	
orb04_1.6	66	66*	66*	66*	66*	66*	78	
orb05_1.6	163	181	181	181	193	174	181	
orb06_1.6	31	31*	31*	31*	31*	31*	**28***	
orb07_1.6	0	0*	0*	0*	0*	0*	0*	
orb08_1.6	621	701	672	672	646	658	669	
orb09_1.6	66	66*	66*	66*	95	66*	83	
orb10_1.6	76	84	84	78	105	97	142	
Total Gap(%)		16.67	15.39	8.18	11.85	6.59	17.38	
Avg. Gap(%)		4.95	4.74	3.12	9.05	2.60	9.01	
# Opt.		15	15	15	14			
# Best Heur.		14	14	16	15	18	14	

† (3,2,2,2,2,2,1,1,1,1)-RF with G/MAI.
‡ (3,2,2,3,2,2,1,1,1,1)-RF with G/MAI.
◇ (3,2,2,3,3,2,1,1,1,1)-RF with G/MAI.

One more performance measure is computed in Table 3 in addition to those in Table 1. For each instance, we determine the best objective value over all heuristics in Table 3, and in rows labeled as "# Best Heur." we report for each category of f the total number of times a given heuristic attains the best solution over 22 instances. It does not make sense to compute the number of times an average objective value is optimal; hence, the entries for LSRW(200), GLS, and BONT are blank in rows "# Opt." For $f = 1.3$, SB-TS dominates both SB(10,3) and LSRW(200) with the computationally less demanding tree parameters (3,2,2,2,2,2,1,1,1,1)-RF and (3,2,2,3,2,2,1,1,1,1)-RF. GLS has a slightly smaller average gap than (3,2,2,3,2,2,1,1,1,1)-RF and (3,2,2,3,3,2,1,1,1,1)-RF; however, both (3,2,2,3,2,2,1,1,1,1)-RF and (3,2,2,3,3,2,1,1,1,1)-RF are better along the other dimensions. The consistency of the performance of SB-TS is clear from the number of optimal solutions achieved and the number of times the best heuristic solution is attained by the SB-TS variants. Analyzing the results for $f = 1.5$, the overall picture is similar. SB-TS with the tree parameters (3,2,2,3,2,2,1,1,1,1)-RF and (3,2,2,3,3,2,1,1,1,1)-RF either performs on a par with or is better than any other existing algorithm for all performance measures. However, note that all algorithms score higher in terms of achieving the best heuristic solution as compared to the results for $f = 1.3$. BONT is clearly inferior to all others, and GLS appears to be better than SB(10,3) and LSRW(200). The best performing algorithm for $f = 1.6$ is LSRW(200) while SB-TS with the tree parameters (3,2,2,3,3,2,1,1,1,1)-RF comes second.

Next, we discuss our results on a new set of benchmark instances created by Essafi et al. (2008) as explained at the beginning of Section 3. Table 4 summarizes the results. The instances la16-la20 are identical to those in previous tables. Except for these 5 instances, the optimal solutions of the instances in Table 4 are unknown, and all performance measures are calculated with respect to the best heuristic solution. Essafi et al. (2008) state that they increase their computational effort significantly for solving this new set of instances; however, it is not possible to deduce the magnitude of the increase from

their paper. For SB-TS, the tree parameters are set to (5,4,3,2,1)-RF for instances with $m = 5$ machines and all other parameters are left intact at their values discussed initially at the beginning of Section 3. However, for instances with $m = 10$ machines we decrease the maximum number of solutions traversed in the neighborhood of the current solution in the tabu search from 60% to 45% of the total size of the neighborhood and fix the tree parameters to (3,2,2,3,2,2,1,1,1,1)-RF in order to avoid long run times. We had to omit the runs with the computationally more intensive tree parameters (3,2,2,3,3,2,1,1,1,1)-RF for instances with more than 100 operations. It appears that this is the limit of the Excel/VB environment. The results in Table 4 reveal that GLS is clearly better than SB-TS for instances with $m = 5$. We reckon that SB-TS cannot realize its full potential for these instances as with a small number of machines selecting a good sequence of scheduling the machines becomes less critical. On the other hand, the performance of SB-TS is enhanced to a significant extent relative to that of GLS for instances with $m = 10$ machines. GLS is superior to SB-TS for $m = 10$ and $f = 1.3$ while SB-TS dominates GLS for $m = 10$ and $f = 1.6$. That is, looser due dates seem to favor SB-TS over GLS.

Table 4: Benchmarking against existing approaches on the instances of Essafi et al. (2008).

			f = 1.3		**f = 1.5**		**f = 1.6**	
Instance	m	n	**SB-TS**†	**GLS**	**SB-TS**†	**GLS**	**SB-TS**†	**GLS**
la01	5	10	2299	2299	1616	1610	1230	1230
la02	5	10	1762	1762	1028	1028	695	695
la03	5	10	1951	1951	1280	1280	1024	1024
la04	5	10	1917	1917	1277	1277	1068	1029
la05	5	10	1878	1878	1205	1205	877	877
la06	5	15	6008	5827	4821	4658	4180	4130
la07	5	15	5961	5801	4624	4548	3843	3988
la08	5	15	5560	5482	4423	4094	3584	3400
la09	5	15	6116	5648	4618	4421	4040	3835
la10	5	15	6734	6621	5304	5148	4728	4533
la11	5	20	12792	12341	10682	10332	9679	9399
la12	5	20	11238	10683	9494	9084	8663	8302
la13	5	20	12533	11889	10158	9846	9244	8916
la14	5	20	13681	13225	12024	11382	11292	10594
la15	5	20	12964	12428	10464	10455	9675	9392
Total Gap(%)			3.65	0.00	3.30	0.00	3.68	0.20
Avg. Gap(%)			2.62	0.00	2.52	0.00	2.71	0.25
# Best Heur.			5	15	4	15	5	14
la16	10	10	1169	1169	166	166	0	0
la17	10	10	899	899	260	260	65	65
la18	10	10	929	929	34	34	0	0
la19	10	10	955	948	23	21	0	0
la20	10	10	805	805	0	0	0	0
la21	10	15	3841	3771	1740	1692	890	949
la22	10	15	4453	4471	2099	2273	1364	1450
la23	10	15	4103	3955	1731	1683	1010	977
la24	10	15	3770	3831	1849	1618	693	773
la25	10	15	3724	3569	1499	1497	929	922
la26	10	20	10562	9748	6328	6106	5236	5125
la27	10	20	9827	9860	6252	6142	4441	4590
la28	10	20	10198	9757	6096	6254	4403	4594
la29	10	20	9792	9397	6265	6392	5049	4706
la30	10	20	9297	8968	5273	5496	3821	4131
Total Gap(%)			3.28	0.16	1.70	1.75	1.80	3.19
Avg. Gap(%)			2.09	0.16	2.34	1.14	0.91	2.69
# Best Heur.			7	12	8	11	11	9

† If $m = 5$, (5,4,3,2,1)-RF with G/MAI.
If $m = 10$, (3,2,2,3,2,2,1,1,1,1)-RF with G/MAI.

Finally, we employ SB-TS to solve 13 instances of $Jm//C_{\max}$ that are classified as "hard" in the literature. See Balas and Vazacopoulos (1998) for a discussion on these instances. Our objective is to demonstrate that SB-TS yields high-quality solutions for the makespan criterion although it is not tailored to this objective and is competitive with an approach specifically developed for $Jm//C_{\max}$. The results are given

in Table 5, where the last two columns "GLS" and "HGA" are taken from Essafi et al. (2008), and the column "HGA" lists the best objective value of the three hybrid genetic algorithms for solving $Jm//C_{\max}$ developed by Goncalves et al. (2005). The tree parameters in SB-TS are specified in the footnotes of Table 5 for each value of m. All other parameters are identical to their previously specified values. Taking into account that the results in column "HGA" are the best over three algorithms, we observe that both GLS and SB-TS outperform HGA although they are not catered toward solving makespan problems.

Table 5: Benchmarking against existing approaches on hard instances of $Jm//C_{\max}$.

Instance	m	n	Opt.	SB-TS[†]	GLS	HGA
la02	5	10	655	655*	655*	655*
la19	10	10	842	842*	842*	842*
la21	10	15	1046	1062	1051	1046*
la24	10	15	935	941	940	953
la25	10	15	977	984	978	986
la27	10	20	1235	1260	1247	1256
la29	10	20	1153[‡]	1190	1174	1196
la36	15	15	1268	1269	1282	1279
la37	15	15	1397	1408	1409	1408
la38	15	15	1196	1202	1196*	1219
la39	15	15	1233	1241	1238	1246
la40	15	15	1222	1234	1230	1241
mt10	10	10	930	936	934	930*
			Total Gap(%)	0.96	0.62	1.19
			Avg. Gap(%)	0.91	0.57	1.11
			# Opt.	2	3	4
			# Best Heur.	4	9	5

[†] If $m = 5$, (4,4,3,2,1)-RF with G/MAI.
If $m = 10$, (3,2,2,3,2,2,1,1,1,1)-RF with G/MAI.
If $m = 15$, (3,2,2,3,2,2,2,1,1,1,1,1,1,1,1)-RF with G/MAI.
[‡] Best known upper bound.

4. Conclusions and Future Research We embedded a tabu search algorithm into the shifting bottleneck algorithm and demonstrated that it results in a state-of-the-art approach for solving JS-TWT. A general insight that may be exploited in order to solve other problems in a job shop setting is that the tree search component of the SB algorithm is a powerful means to diversify the embedded local search. Moreover, by controlling the size of the search tree we can easily trade-off solution quality and time. The tabu search component of our solution algorithm relies on two distinct neighborhood definitions with complementary properties and the demonstrated synergy merits further research in this area. Moreover, leveraging on previous research by Balas and Vazacopoulos (1998) and Kreipl (2000) we focus on neighborhoods that generalize adjacent pairwise interchanges on a critical path and conclude that more work in this area may be fruitful for job shop scheduling problems with regular objectives.

Another significant contribution of our work is a novel approach for solving a generalized single-machine weighted tardiness problem that arises as a subproblem in the SB heuristic. Our approach is based on a preemptive lower bound that provides information for constructing non-preemptive feasible solutions. Although the results are satisfactory there is room for improvement as we argued in Section 2.2. We leave this issue as a future research direction.

One hurdle that we would need to overcome in order to be able to solve larger instances of JS-TWT by SB-TS is reducing the solution times. An obvious direction here is to replace the exact longest path calculations from scratch for each move in the neighborhood of the current solution in the tabu search by approximate move evaluations. More work is needed on approximate move evaluations in the MAI and G/MAI neighborhoods in the context of JS-TWT. Other options for shorter solution times involve parallelizing SB-TS as different branches of the search tree may be pursued independently and applying warm starts to similar transportation problems solved during the course of the algorithm.

Finally, we emphasize the need for new optimal algorithms capable of solving large instances of JS-TWT. To date, there is a single paper available in this domain and new research in this area would be most welcome.

*Bibliography

Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401.

Armentano, V. A. and Scrich, C. R. (2000). Tabu search for minimizing total tardiness in a job shop. *International Journal of Production Economics*, 63(2):131–140.

Aytug, H., Kempf, K., and Uzsoy, R. (2002). Measures of subproblem criticality in decomposition algorithms for shop scheduling. *International Journal of Production Research*, 41(5):865–882.

Balas, E., Lenstra, J. K., and Vazacopoulos, A. (1995). The one-machine problem with delayed precedence constraints and its use in job shop scheduling. *Management Science*, 41(1):94–109.

Balas, E. and Vazacopoulos, A. (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2):262–275.

Bulbul, K., Kaminsky, P., and Yano, C. (2007). Preemption in single machine earliness/tardiness scheduling. *Journal of Scheduling*, 10(4-5):271–292.

Dauzere-Peres, S. and Lasserre, J. (1993). A modified shifting bottleneck procedure for job shop scheduling. *International Journal of Production Research*, 31(4):923–932.

De Bontridder, K. (2005). Minimizing total weighted tardiness in a generalized job shop. *Journal of Scheduling*, 8(6):479–496.

Demirkol, E., Mehta, S., and Uzsoy, R. (1997). A computational study of shifting bottleneck procedures for shop scheduling problems. *Journal of Heuristics*, 3(2):111–137.

Dyer, M. and Wolsey, L. (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26(2–3):255–270.

Essafi, I., Mati, Y., and Dauzere-Peres, S. (2008). A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research*, 35(8):2599–2616.

Goncalves, J. F., de Magalhaes Mendes, J. J., and Resende, M. G. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operations Research*, 167(1):77–95.

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

Jain, A. and Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operations Research*, 113(2):390–434.

Kreipl, S. (2000). A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3):125–138.

Lawrence, S. (1984). Supplement to "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques". GSIA, Carnegie Mellon University, Pittsburgh, PA.

Lenstra, J., Rinnooy Kan, A., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.

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

Matsuo, H., Suh, C., and Sullivan, R. (1988). A controlled simulated annealing method for the general job shop scheduling problem. Technical Report 03-04-88, Department of Management, Graduate School of Business, The University of Texas at Austin, Austin, TX.

Mattfeld, D. and Bierwirth, C. (2004). An efficient genetic algorithm for job shop scheduling with tardiness objectives. *European Journal of Operations Research*, 155(3):616–630.

Pinedo, M. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer, 3rd edition.

Pinedo, M. and Singer, M. (1999). A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, 46(1):1–17.

Roy, B. and Sussman, B. (1964). Les problemes d'ordonnancement avec contraintes disjonctives. Paris: Note DS No.9 bis. SEMA.

Runge, N. and Sourd, F. (2009). A new model for the preemptive earliness-tardiness scheduling problem. *Computers & Operations Research*, 36(7):2242–2249.

Singer, M. (2001). Decomposition methods for large job shops. *Computers & Operations Research*, 28(3):193–207.

Singer, M. and Pinedo, M. (1998). A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions*, 30(2):109–118.

Standard Performance Evaluation Corporation (1996). SPEC CINT95 Benchmarks. http://www.spec.org/cgi-bin/osgresults?conf=cint95. Accessed in February 2010.

Standard Performance Evaluation Corporation (2007). SPECint2006 Results. http://www.spec.org/cgi-bin/osgresults?conf=cint2006. Accessed in February 2010.

Suh, C. (1988). *Controlled search simulated annealing for job shop scheduling*. PhD thesis, University of Texas, Austin.

Vepsalainen, A. P. J. (1987). Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):1035–1047.