Int J Adv Manuf Technol (2007) 34: 378–384
DOI 10.1007/s00170-006-0590-7

ORIGINAL ARTICLE

Bibo Yang

Single machine rescheduling with new jobs arrivals and processing time compression

Received: 8 January 2004 / Accepted: 8 August 2005 / Published online: 1 June 2006

Abstract We consider job rescheduling problems where rescheduling is required in response to newly arriving jobs. To reduce the negative impacts of disruptions to the original schedule, the processing times of the newly arriving jobs can be reduced at a cost, which we call a time compression cost. The objective of the problem is to minimize total cost after rescheduling, which includes schedule disruption costs, time compression related costs, and a cost that depends on a traditional measure of schedule efficiency. We separately consider two different measures of schedule efficiency: total completion time and weighted tardiness, and present polynomial time algorithms for the total completion time case. For the weighted tardiness cost efficiency measure, we provide a heuristic based on very large scale neighborhood (VLSN) search.

Keywords Rescheduling new job disruption compression

1 Introduction and problem definition

This paper considers rescheduling problems that arise due to new job arrivals subsequent to developing a planned execution schedule that consists of a set of original jobs. During processing of the original jobs, new orders (jobs) may arrive to the system. The scheduling firm wishes to reschedule its set of remaining jobs (including new jobs) in order to best integrate the new jobs into the previously planned schedule. In order to reduce the disruption on the schedule, as is the case in many practical scheduling contexts, the firm can pay an additional premium to reduce the processing time (through, for example, overtime or subcontracting) of the new jobs. This situation occurs in many make-to-order (MTO) environments, where emergency orders come in occasionally and the firm can use additional resources to complete the emergency orders.

B. Yang (✉)
Industrial and Systems Engineering Department,
University of Florida,
Gainesville, FL 32611, USA
e-mail: yangbibo@ufl.edu
Tel.: +1-352-3927434
Fax: +1-352-3923537

We assume that the firm utilizes a primary measure of efficiency in scheduling jobs, such as the minimum total flow time or minimum weighted tardiness. In addition to this primary efficiency measure, the firm is also concerned about disruption costs incurred as a result of schedule changes in response to newly arriving jobs.

The problems we consider are single machine rescheduling problems with compression time and without preemption, which are defined as follows. An original schedule exists for some n_O original jobs. Each job i has associated parameters including a processing time p_i, due date d_i, and tardiness cost l_i (per unit time tardy). Suppose that at the beginning of processing the planned schedule for the n_O original jobs, n_N new jobs arrive. The total number of jobs equals $n_O + n_N = n$. In addition to the parameters described for the original jobs, each newly arriving job i has a maximum compression time u_i, and a unit time compression cost of c_i. This compression time and associated cost represent the possibility of reducing the required processing time of a job at some cost per unit time reduction. The processing time of job i can therefore be reduced by x_i ($\leq u_i$) units of time at a compression cost equal to $c_i x_i$. We must alter the original schedule in order to account for processing the new jobs. A rescheduling, or disruption cost is associated with the degree of deviation from the initial schedule. This disruption cost accounts for administrative costs and other coordination costs associated with supplier and customer disruptions. We use the absolute change in the originally scheduled starting time of a job as the measure of schedule deviation. We define a unit disruption cost of h, i.e., when a job's starting time is changed from s_i to s_i', the disruption cost equals $h\left|s_i - s_i'\right|$.

Our objective is to obtain an efficient and stable schedule; we therefore wish to minimize a composite objective consisting of some traditional scheduling objective (e.g., total weighted tardiness cost or total flow time), plus schedule disruption costs and new job processing time compression costs.The remainder of this paper is organized as follows. Section 2 next discusses related past work on rescheduling problems and its relation to our work. Sections 3 and 4 consider the rescheduling problem under different efficiency performance measures. These sections present results and solution methods when the efficiency measure component of the objective function is the minimum total completion time (Section 3) and the minimum weighted tardiness cost (Section 4) respectively. Section 5 then summarizes and provides potential directions for future research.

2 Literature review

Any *rescheduling* problem must simultaneously address at least two criteria. One is the efficiency of the schedule, which is a traditional scheduling problem approach, using an objective such as minimum makespan, minimum weighted completion time, minimum total tardiness cost, minimum number of tardy jobs, etc. The second criterion is schedule stability, which refers to the degree of deviation from an original schedule. The measure of stability may be some measure of the change in completion times, the change in job sequence, or the change in the starting times, which we use in our analysis. A large number of rescheduling research problems may be explored through selecting and modeling these two criteria. We next discuss relevant past work that addresses such problems.

Norbis and Smith [1] study a rescheduling problem with resource constraints, where disruption events are defined as changes in resource availability, changes in due dates, changes in processing times, or new job arrivals. They present a multi-objective mathematical programming formulation for the resource scheduling problem. A quasi-dynamic approach, based on periodic data updating and re-optimizing is presented to take control whenever a disruption event occurs. If some resource availability is violated, the procedure may delay operations with low priority and release the associated resources for work on higher priority jobs.

Bean and Birge [2] considered a rescheduling problem with machine disruptions, and investigated “match-up” scheduling heuristics, which compute a transient schedule after a machine disruption. When a disruption occurs, their approach determines how to merge back into the pre-planned schedule at some future time. This approach first fixes an initial *match up* point, and resequences jobs on disrupted machines to minimize total tardiness cost. If the cost for meeting the selected match-up point exceeds a predetermined threshold, the match-up point is then incremented by some value, until the match-up point reaches a predetermined maximum value. A match-up is therefore possible only if there is enough idle time existing in the original schedule.

Wu, Storer et al. [3] proposed a rescheduling procedure that generates a new schedule at each occurrence of a shop disruption. They use a bicriterion approach, where the two conflicting objectives are minimizing makespan and minimizing deviation from the original schedule. Two sets of local search heuristics were developed; the first set used pairwise swapping methods, using a weighted combination of the two objectives to create a single objective. The second is a local search heuristic based on a genetic algorithm approach, considering a two dimensional space of makespan and deviation; the quality of any given solution point in the space can be measured based on its Euclidian distance to the origin.

Leon, Wu et al. [4] consider robustness measures and robust scheduling methods for job-shop rescheduling with fallible machines. Robustness is expressed as a linear combination of the actual makespan after a disruption and the deviation of the new makespan from the previous value. A time window called *slack* is defined for each operation, within which the operation can be started without incurring any makespan delay. The robustness measure is the average slack time of the operations to be processed on fallible-machines. A robust scheduling solution based on genetic algorithms is proposed.

Unal, Uzsoy et al. [5] consider a single machine scheduling problem with newly arriving jobs that have setup times that depend on their part types. They consider inserting new jobs into the original schedule so as to minimize the total weighted completion time or makespan of the new job. Their approach does not, however, incorporate a measure of schedule disruption.

Hall and Potts [6] consider a rescheduling problem with newly arriving orders. They present two classes of models. In the first class they minimize schedule cost subject to a limit onthe deviation from the original schedule. In the second class, they minimize a total cost objective, which includes both the original cost measure and the cost of deviation. The efficiency measures they use are either maximum lateness or total completion time, while the stability measures are the maximum (ortotal) sequence disruption or maximum (or total) completion time deviation. For each problem, with the performance measures defined above, they either present a polynomial algorithm or prove that one does not exist.

The problem we consider is similar to that of Hall and Potts [6], since we consider new job arrival disruptions, and use a total cost objective function. In addition to the simple efficiency measure used in their paper (total flow time), we also consider a more complex efficiency measure, which is the total weighted tardiness cost. We also consider the added feature of available compression time, which allows reducing the new job’s processing times to decrease the impact of disruptions on the schedule, which to the best of our knowledge is a new approach in rescheduling research.

3 Efficiency measure 1: minimizing total completion time

In this section, we consider the rescheduling problem with an objective function of minimizing the sum of total completion time, compression cost of the new jobs and the disruption costs incurred as a result of starting time changes for the original jobs. We use the following notation.

Notation

C_i	the completion time of job j.
J_O	The set of n_O old jobs.
J_N	The set of n_N new jobs.
π^*	an optimal schedule of the old jobs.
$\Delta_j(\pi^*)$	the deviation between starting times of job j in schedules π^* and the new schedule.

In terms of standard scheduling notation, we denote this problem as $1\|\sum C_j + \sum c_j x_j + h\sum \Delta_j(\pi*)$. We know that the shortest processing time (SPT) rule provides the optimal sequence for the problem $1 \| \sum C_j$. Thus, we assume that in schedule π^*, the old jobs were sequenced by SPT rule.

Hall and Potts [6] presented a lemma for the rescheduling problem $1\|\sum C_j + h\sum \Delta_j(\pi*)$ (the rescheduling problem to minimize total completion time and disruption costs). The lemma states that there exists an optimal schedule in which the jobs in the set J_O are sequenced in SPT order as in π^*. In their problem, the processing times of the newly arriving jobs are fixed; in our problem, we allow for compressing processing times for the new jobs to reduce schedule disruption impacts. However, this lemma is still valid with the added dimension of processing time compression since the SPT order only depends on the processing times of old jobs.

Lemma 1 For the rescheduling problem $1\|\sum C_j + \sum c_j x_j + h\sum \Delta_j(\pi*)$., there exists an optimal schedule in which jobs in the set J_O are sequenced in SPT order as in π^*.

Let us next consider another related problem, the problem of minimizing total completion time problem with compression costs, i.e., $1\|\sum C_j + \sum c_j x_j$. This problem has been studied by Vickson [7]. Let job [j] denote the job in position j. He shows that given a compression time vector x, the cost (including the total flow time and compression cost) is given by:

$$K(x) = \sum_{j=1}^{n} \left[(n-j+1)p_{[j]} + \left(c_{[j]} - (n-j+1)\right)x_{[j]}\right]$$

This function achieves its minimum with respect to the [j]th component of x at $x_{[j]}=u_{[j]}$ if$c_{[j]}<n-j+1$, and at $x_{[j]}=0$ otherwise. Thus the cost incurred by placing job i in position j, and then selecting its processing time optimally, does not depend on the jobs which precede or follow it in the sequence, but only on the sequence position. Therefore, the problem can be formulated as an assignment problem with a cost k_{ij} for assigning job i to position j given by $k_{ij} = (n-j+1)p_i + \min\{0, (c_i - (n-j+1))u_i\}$, which can be solved in $O(n^{2.5})$ time.

We can create an assignment-based algorithm for the rescheduling problem that uses the aforementioned theorem by Vickson [7]. The idea is simple: we insert the new jobs into the initial schedule of the old jobs, which were sequenced by the SPT rule. Given a new job i, suppose we insert it into position j of the new schedule, and assume that there are m old jobs after job i. The cost incurred includes the increased flow time, the compression cost and the disruption cost, which is given by:

$$\begin{aligned} k_{ij,m} &= (n-j+1)(p_i - x_i) + c_i x_i + mh(p_i - x_i) \\ &= (n-j+1+mh)p_i + (c_i - (n-j+1) - mh)x_i \\ &= (n-j+1+mh)p_i \\ &\quad + \min\{0, (c_i - (n-j+1) - mh)u_i\} \end{aligned}$$

Based on this, we set $x_i = u_i$ if $ci<n-j+1+mh$, and set $x_i = 0$, otherwise. Thus, given a new job, the "insertion" cost does not depend on which jobs precede or follow it in the sequence but only on the values i,j and m. Therefore, the problem can also be formulated as an assignment problem with assignment cost $k_{ij,m}$.

For each possible $m = 1, \ldots, n_O$, there are at most n_N positions j for job i and there are $O(n_O n_N)$ values of $k_{ij,m}$. Thus the assignment problem can be solved by $O({n_0}^{2.5}{n_N}^{2.5})$ time. Solving this assignment problem then provides the optimal solution for this rescheduling problem.

4 Efficiency measure 2: minimum total tardiness cost

In the previous section, we discuss a rescheduling problem which is solvable in polynomial time. In this section, we consider a NP-Hard rescheduling problem. We provide a heuristic solution method using very large scale neighborhood (VLSN) search methods. This heuristic can be used to solve many NP-Hard rescheduling problems with different objective functions (such as largest lateness with total disruption cost and many others). As an example of the application of this heuristic method, we consider a rescheduling problem with an objective of minimizing the sum of total weighted tardiness cost, compression cost, and disruption cost. The tardiness cost for job i is $(C_i - d_i)^+ l_i$, where $(a)^+ = \max\{a, 0\}$. In standard scheduling notation terms, we denote this problem as $1 \| \sum l_j (C_j - d_j)^+ + \sum c_j x_j + h\sum \Delta_j(\pi)$. We know that the minimum weighted tardiness cost *rescheduling* problem is NP-Hard because the corresponding single-machine scheduling problem is NP-Hard [8]. Therefore, this more general problem with compression and disruption costs is also NP-Hard.

4.1 A Heuristic based on VLSN

This heuristic solves problem $1|| \sum l_j(C_j - d_j)^+ + \sum c_j x_j + h\sum \Delta_j(\pi)$ by a very large scale neighborhood search, with a starting solution that uses the same sequence for the old jobs as in the original schedule.

Given a feasible solution (a schedule), its neighborhood is generated by changing the sequence of jobs and changing the compression time applied to each new job. A larger search neighborhood typically implies better quality of locally optimal solutions and of the final solution. At the same time, the larger the neighborhood, the longer it takes to search the neighborhood at each iteration. For this reason, a larger neighborhood does not necessarily produce a more effective heuristic, unless one can search the larger neighborhood in a very efficient manner. There are a number of neighborhood search heuristics in the literature applied to a broad range of combinatorial optimization problems. Here we will use a technique called VLSN (very large-scale neighborhood search); see, for example, Ahuja, Ergun et al. [9]. Although this heuristic approach has been used to solve a number of classical NP-hard problems, to our knowledge, this is the first application to a rescheduling problem.

In VLSN, the neighborhood is searched using network flow techniques that implicitly enumerate solutions in the neighborhood. Given an initial schedule, we construct an improvement graph that allows us to search a pre-defined large neighborhood structure. Each arc in this graph represents an alteration of the schedule, and has an associated cost determined by the nature of the change. An improved schedule is found by finding a shortest path in the improvement graph with negative cost, which implies that a lower cost neighboring solution exists. We next explain how we construct such an improvement graph.

Let $G(V, E)$ be the improvement graph associated with the current schedule. Then the node set $V = \{0, 1, 2, 3, \ldots, n, n+1, 1', 2', \ldots n_N'\}$ of G consists of $n+n_N+2$ elements, where each node i represents the job in position $[i]$ for $1 \le i \le n$; nodes 0 and $n+1$ are dummy nodes; node i' represents a new job i for $1 \le i \le n_N$. An arc (i, j) in the graph represents a move involving jobs $(i, \ldots, j-1)$, for $1 \le i, j \le n+1$. An arc ending at node j' represents a change in compression time for the new job j for $1 \le j \le n_N$. The arc set E consists of all the arc types that we next define. The arc set E is defined and created as follows.

Arcs $(i, i+1)$ for $1 \le i \le n$: these arcs represent the situation where job $[i]$ is not involved in any move. Hence the cost of these arcs is: $c(i,i+1) = 0$.

Arcs (i, j) for $1 \le i \le n-1$ and $i+2 \le j \le n+1$: these arcs represent local updates that only affect jobs $[i]$, ..., $[j-1]$. Let$s^0_{[i]}$ denote the starting time of job $[i]$ in the original schedule, while $C_{[i]}$, $s_{[i]}$, and $p_{[i]}$ are the current completion, starting, and processing times in the revised schedule (recording in the node i). There are three of these kinds of arcs in the graph or each possible (i, j) pair:

Insertions arcs $(i, j)^1$ represent the ejection of job $[i]$ from its current position and its re-insertion between job $[j-1]$ and job$[j]$. The cost for these arcs is as follows:

$$\begin{aligned} C(i,j)^1 = {} & l_{[i]}\left[\left(C_{[i]} + \sum_{k=i+1}^{j-1} p_{[k]} - d_{[i]}\right)^+ - \left(C_{[i]} - d_{[i]}\right)^+\right] \\ & + \sum_{k=i+1}^{j-1} l_{[k]}\left[\left(C_{[k]} - p_{[i]} - d_{[k]}\right)^+ - \left(C_{[k]} - d_{[k]}\right)^+\right] \\ & + h\left|s_{[i]} + \sum_{k=i+1}^{j-1} p_{[k]} - s^0_{[i]}\right| - h\left|s_{[i]} - s^0_{[i]}\right| \\ & + \sum_{k=i+1}^{j-1} h\left(\left|s_{[k]} - p_{[i]} - s^0_k\right| - \left|s_{[k]} - s^0_{[k]}\right|\right) \end{aligned}$$

The first and the second terms in the function are the increase in tardiness cost of job $[i]$ and jobs $[i+1]$, ..., $[j-1]$, while the third and fourth terms are the increase in disruption costs of job $[i]$ and jobs $[i+1]$, ..., $[j-1]$.

Swaps Arcs $(i, j)^2$ represents the swap of job $[i]$ with job $[j-1]$. The cost for these arcs is as follows:

$$\begin{aligned} C(i,j)^2 = {} & l_{[i]}\left[\left(C_{[i]} + \sum_{k=i+1}^{j-1} p_{[k]} - d_{[i]}\right)^+ - \left(C_{[i]} - d_{[i]}\right)^+\right] + l_{[j-1]}\left[\left(C_{[j-1]} - \sum_{k=i}^{j-2} p_{[k]} - d_{[j-1]}\right)^+ - \left(C_{[j-1]} - d_{[j-1]}\right)^+\right] \\ & + \sum_{k=i+1}^{j-2} l_{[k]}\left[\left(C_{[k]} - p_{[i]} + p_{[j-1]} - d_{[k]}\right)^+ - \left(C_{[k]} - d_{[k]}\right)^+\right] \\ & + h\left|s_{[i]} + \sum_{k=i+1}^{j-1} p_{[k]} - s^0_{[i]}\right| - h\left|s_{[i]} - s^0_{[i]}\right| + h\left|s_{[j-1]} - \sum_{k=i}^{j-2} p_{[k]} - s^0_{[j-1]}\right| - h\left|s_{[j-1]} - s^0_{[j-1]}\right| \\ & + \sum_{k=i+1}^{j-2} h\left(\left|s_{[k]} - p_{[i]} + p_{[j-1]} - s^0_{[k]}\right| - \left|s_{[k]} - s^0_{[k]}\right|\right) \end{aligned}$$

The first through third terms in $C(i,j)^2$ represent the increase in tardiness cost of jobs [i], ..., [j−1] and [i+1],..., [j−2], while the fourth through sixth terms are the increase in disruption costs for jobs [i], ..., [j−1] and [i+1], ..., [j−2].

2-opts Arc $(i, j)^3$ represents a 2-opt move involving the subsequence of the jobs [i], [i+1],...[j−1] in the current schedule, i.e., the processing order of the jobs [i], [i+1], ..., [j − 1] is reversed. Let $C'_{[l]}$ denote the completion time of job [l] after such a reversal, where $C'_{[l]} = C_{[l]} + \sum_{q=l+1}^{j-1} p_{[q]} - \sum_{q=i}^{l-1} p_{[q]}$, for 1 = i, ..., j − 1. The cost for such arcs is then given by

$$C(i,j)^3 = \sum_{k=i}^{j-1} l_{[k]} \left[\left(C'_{[k]} - d_{[k]} \right)^+ - \left(C_{[k]} - d_{[k]} \right)^+ \right] + \sum_{k=i}^{j-1} h\left(\left| C'_{[k]} - p_{[k]} - s^0_{[k]} \right| - \left| s_{[k]} - s^0_{[k]} \right| \right).$$

Compression Arc $(i, j')^y$ (where $0 \le i \le n+1$) or arc$(i', j')^y$ (where $1 \le i < n_N$) represents the action of adding y units of compression time to the new job j. If new job j in the current schedule has x_j units of compression time, the graph will have (u_j +1) copies of $(i, j')^y$ or arc $(i', j')^y$, with $y = -x_j, ..., 0, ..., (u_j - x_j)$. If the new job is in position [k], the cost for these arcs is as follows:

$$C(i,j)^y = \sum_{q=k}^{n} l_{[q]} \left[\left(C_{[q]} - y - d_{[q]} \right)^+ - \left(C_{[q]} - d_{[q]} \right)^+ \right] + \sum_{q=k+1}^{n} h\left(\left|{}_{[q]} - y - s^0_{[q]} \right| - \left| s_{[q]} - s^0_{[q]} \right| \right) + c_j y,$$

where $C_{[q]}$, $s_{[q]}$, and $p_{[q]}$ are the completion, starting, and processing times of job [q]respectively, in the current schedule. The first term in the above expression is the change in tardiness cost, the second term is the disruption cost, while the third term is the new job's compression cost.

As Fig. 1 shows, the improvement graph is acyclic. An improved schedule is obtained by finding a path from node 0 to node n_N', where the sum of costs of the arcs in the path is negative.

Theorem 1 An algorithm exists that searches the defined neighborhood in $O(n^3)$.

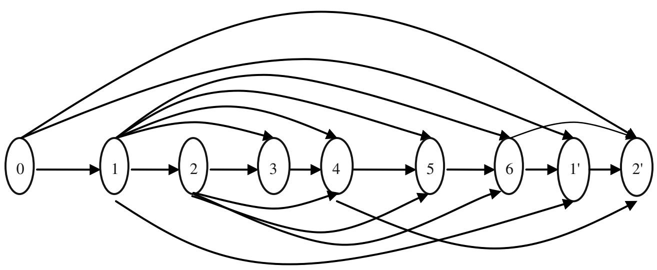

Fig. 1 An example of the improvement graph

Proof Since the graph is acyclic, a distance label correcting algorithm solves the shortest path problem in $O(m)$ time, where m is the number of edges. There are $O(n)$ arcs for each kind of move from a node, which implies that there are $O(n^2)$ arcs in the graph. At each step, the calculation of the cost of arc (i,j) requires $O(n)$. Therefore, the total complexity for searching the neighborhood is $O(n^3)$.

Notice that the cost of arc (i, j) not only depends on nodes i and j, but also on the path leading to node i, because the schedule may be changed after traversing each arc. If we want to find the best improvement, we must therefore enumerate all paths to every node, which implies exponential complexity.

As a heuristic approach to overcome this, in our label correcting algorithm, we express the state of the schedule at each node by the completion time and process time of all jobs; we keep this information updated at each step (at each node). When we come to node i, we check all incoming arcs to the node and select the arc (j, i) that produces the lowest node i distance label, and let the distance label of node i equal the distance label of node j plus the cost of arc (j, i). We then update the completion time for all jobs and the process time for the new jobs (thus update the compression time) according to the move taken, and update the cost of each arc leaving this node. Note that for each node, we only select one incoming arc to the node, and we only check a subset of paths to each node. Therefore, the heuristic algorithmwe apply does not guarantee obtaining the best improvement possible in the neighborhood for the current schedule. In other words, if no negative shortest path exists at an iteration, this does not mean there is no improving schedule in the neighborhood of current schedule.

At each iteration, the algorithm constructs an improvement graph and finds the shortest path in the graph. If the shortest path has positive cost, we stop the iteration; otherwise, we update the current schedule, construct another improvement graph, and continue the search. To improve the solution procedure, we use multiple starting points. An initial solution is constructed by simply inserting the new jobs in some positions in the old schedule while keeping the sequence of all other jobs fixed. To obtain multiple starting points, we construct several initial schedules where the new jobs inserted randomly in different positions.

As we mentioned before, this heuristic may be applied to other rescheduling problems with different objective functions. The construction procedure and the search method of the improvement graph are the same. We only need modify the definition of the cost of arcs in the graph.

4.2 Branch and bound algorithm

To evaluate the efficiency of the VLSN heuristic, we compare its performance to the optimal solution obtained via a customized branch and bound procedure. The branch and bound algorithm begins with an empty schedule and adds a job into the schedule at each level k, for k=1, ...,n, where a level corresponds to the numberof jobs added to create a partial schedule. At each node at level k, a job is added in the $(n - k + 1)$st position in the schedule, and a lower bound on cost at this node and its children nodes is computed as the cost of the partial schedule. Note that this implies that we add jobs to the schedule in reverse order as we move down the tree. The algorithm uses a depth-first strategy.

When we add a new job, we always use its minimum process time and ignore its compression cost. Thus we always get a lower bound on cost for the partial schedule at any node. Only when we reach a leaf node, where all the jobs have been scheduled, do we enumerate all possible values of compression time for the new jobs, calculate the total cost (including the compression cost at that time), and select the best as the cost for this leaf node. During the search, if the lower bound of a node is larger than the current upper bound, we prune the node. The initial upper bound is the cost of the solution from the VLSN heuristic; this value is updated whenever we find a better feasible solution.

4.3 Computational testing

We randomly generated two problem classes, with each problem instance containing 27 old jobs and 3 new jobs. We chose relatively small problem sizes so that we can compare the exact branch and bound algorithm solution to the heuristic results in reasonable computing time. The first problem class uses an optimal initial schedule for the original set of jobs, while the second problem class does not guarantee the initial schedule is optimal. In the first class of problems, the optimal schedule after rescheduling is expected to be similar to the original one. Thus it is easier for VLSN to perform the neighborhood search than for the second class of problems, where we may require a great deal of neighborhood search. The test problem parameters are generated using the following rules,where U[a, b] denotes the uniform distribution between a and b.

- Disruption cost per unit time, h = U[1, 3];
- Job processing times, p_i=U[3, 12], for $1 \leq i \leq n$;
- Original schedule completion times, $C_i = C_{i-1} + p_i$, for $1 \leq i \leq n_O$;
- Job tardiness cost per unit time, l_i=U[1, 2h], for $1 \leq i \leq n$;
- New jobs due date, $d_i = U\left[\sum_{i=1}^{n} p_i\right]$, for $1 \leq i \leq n_N$.
- Compression time upper bound, u_i= U[1, 3], for $1 \leq i \leq n_N$.

For the first problem class, we set $d_i = C_i$+U[0, 3], while for the second problem class, we set $d_i = C_i$+U[−4, 4], for $1 \leq i \leq n_O$. For each problem class we generated 50 test problem instances.

Since the branch and bound algorithm may run quite long for some problem instances, we limit the branch and bound solution time to one hour, and use the best upper bound obtained by the branch and bound algorithm as a benchmark for comparison to the VLSN approach. For the VLSN heuristic, for each instance, we randomly generated 15 initial solutions by simply inserting new jobs into the old schedule. Table 1 summarizes the results of our computational tests. The average error is taken as the average difference between the VLSN solution and the branch and bound (B&B) solution values, divided by the B&B solution value.

The table shows that VLSN is able to obtain very good solution quality in a much shorter time than branch and bound. Note that when we ran the branch and bound algorithm, we used the results from VLSN as the initial lower bound.Without this lower bound, branch and bound is expected to produce worse results. We also found that if we only increase the problem size to 40 jobs, the branch and bound algorithm is truncated in one hour for almost all the instances. Therefore, the VLSN heuristic is a fast and efficient method for solving the rescheduling problem with newly arriving jobs.

As expected, we also observe from the tables that both VLSN and B&B do much better in problem class one than in class two. B&B has the same worst case complexity for the two problem classes, since it can, in the worst case, enumerate all possible solutions. However, B&B uses the solution of VLSN as the original upper bound, and this upper bound is generally closer to the optimal solution for problem class 1, since the original schedule is optimal for this problem class. This leads to better B&B performance for problem class 1. This suggests that although VLSN is an efficient method for rescheduling problems given a good initial schedule for the existing old jobs, it may not be as efficient for other scheduling problems.

5 Conclusions and future research

This paper examines the rescheduling problem for newly arriving jobs with compression time, where there are several competing objectives. We measure the stability of the schedule using a rescheduling cost. The efficiency of the schedule is evaluated using two types of efficiency measures. The objective function is the sum of the rescheduling cost and the efficiency cost; thus we transform there scheduling problem into a minimum cost scheduling problem with a single objective. For the first efficiency measures, we provided polynomial-time algorithms. For the second efficiency measure, which leads to an NP-hard optimization problem, we implemented a VLSN search heuristic. We provided results of computational tests based on randomly generated problem instances and showed that the VLSN heuristic provides high-quality solutions in very fast time.

Table 1 Results for the computational test

	Problem class 1	Problem class 2
Average error	0.44%	0.84%
# instances B&B are truncated	15	18
# instances VLSN gets optimal solution	32	26
# instances B&B gets better solution	5	5
Avg. run time for VLSN (min.)	0.04	0.07
Avg. run time for B&B (min.)	17	25.4

Future work in this area might consider stochastic models for cases in which job arrivals follow some probability distribution, as well as other stability measures such as a measure of the deviation of the sequence. Since rescheduling may be motivated by many other factors, such as machine failures, it may also be interesting to implement our methods for these rescheduling problems as well.

References

1. Norbis MI, Smith JM (1988) A multi objective, multi-level heuristic for dynamic resource constrained scheduling problems. Eur J Oper Res33:30–41
2. Bean JC, Birge JR,Mittenthal J, Noon CE (1991) Match-up scheduling with multiple resources,release dates and disruptions. Oper Res 39:470–483
3. Wu SD, Storer RH, ChangPC (1993) One-machine rescheduling heuristics with efficiency and stability as criteria. Comp Oper Res 20:1–14
4. Leon, VJ, Wu SD, StorerRH (1994) Robustness measures and robust scheduling for job shops. IIE Trans26:32–43
5. Unal AT, Uzsoy R, Kiran AS (1997) Rescheduling on a single machine with part-type dependent setup times and deadlines. Ann Oper Res 70:93–113
6. Hall NG, Potts CN (2004) Rescheduling for new orders. Oper Res 52(3):440–453
7. Vickson RG (1980) Choosing the job sequence and processing times to minimize total processing plus flow cost on a single machine. Oper Res 28(5):1115–1167
8. Du J, Leung JY-T (1990) Minimizing total tardiness on one machine is NP-hard. Math Oper Res 15:483–495
9. Ahuja RK, Ergun O, Orlin JB, Punnen AP (2002) A survey of very large-scale neighborhood search techniques. Discret Appl Math 123:75–102