Proceedings CPAIOR'02

Combining Local Search and Linear Programming to Solve Earliness/Tardiness Scheduling Problems

J. Christopher Beck
ILOG, SA
9, rue de Verdun, BP85,
94253 Gentilly Cedex, France
email: cbeck@ilog.fr

Philippe Refalo
ILOG, SA
Les Taissounières, 1681, route des Dolines
Sophia-Antipolis 06560 VALBONNE, France
email: refalo@ilog.fr

Abstract

The problem of scheduling multiple jobs on multiple unary capacity machines is addressed in the case where each job has a convex piecewise linear cost function representing earliness and tardiness costs for a given due date. The solution approach is a hybrid of tabu search used to find a sequence of activities on each machine and linear programming used to assign optimal start times to each activity given the sequence. Experimental results indicate that in many cases this approach is able to find equal or better solutions than approaches based on constructive hybrid techniques or mixed integer programming.

1 Introduction

Scheduling with due dates and earliness and tardiness costs is particularly relevant to real world scheduling problems. Ideally, every process will end exactly on its due date. A tardy process (i.e., one that ends after its due date) has negative implications for customer satisfaction, time to market, and smooth operation of supply chains. Conversely, an early process (i.e., one that ends before its due date) requires increased inventory storage and

capital investment while degrading the agility of the enterprise to react to changes in the marketplace.

Previous work on scheduling with earliness and tardiness costs [2] used a hybrid constraint and linear programming approach in a constructive framework building on probe backtracking [5]. In this paper, we address such problems using a combination of local search and linear programming. A tabu-based local search technique defines a neighborhood based on swapping consecutive activities that are both on the same resource and on a critical path of the current solution. Given a sequence, an associated linear programming solver computes the optimal start time for each activity and the cost of the solution. In other words, for each possible neighbor of a solution a linear program is solved. Decisions about moving to a new solution are made on the basis of the cost from the LP solver.

2 Problem Definition

An $n \times m$ earliness/tardiness scheduling problem (ETSP) consists of a set, J, of n jobs, m activities per job, and a set, R, of k resources. For each job j there is an ordered set of activities $A(j) = \{a_{j1}, a_{j2}, \ldots, a_{jm}\}$. Each activity is characterized by its start time $s(a_{ji})$ and its processing time $pt(a_{ji})$. Let $s_{min}(a_{ji})$, represent the minimum start time of activity a_{ji} and $s_{max}(a_{ji})$ represent its maximum start time. No pre-emption is allowed, meaning that once an activity has started execution, it must execute for its entire duration without interruption.

For each job, $j \in J$, there is a due date, dd_j, and two cost factors, earliness, ec_j, and tardiness, tc_j. Earliness is the cost factor used if the last activity a_{jm} of job j ends before its due date while tardiness is the cost factor used if it ends after its due date. More precisely the cost incurred for each job j is:

$$c_j = \begin{cases} ec_j(dd_j - l_j) & \text{if } l_j \leq dd_j \\ tc_j(l_j - dd_j) & \text{if } l_j > dd_j \end{cases} \tag{1}$$

Where l_j is the completion date of last activity a_{jm}. That is, $l_j = s(a_{jm}) + pt(a_{jm})$. Observe that this cost is 0 if the job finishes on time, that is, if $l_j = dd_j$.

The cost for the entire problem, c_J, is the sum of the costs incurred for each job, $j \in J$:

$$c_J = \sum_{j \in J} c_j \tag{2}$$

In the restricted version of the ETSP, each activity, a_{ji}, requires the use of a single unary capacity resource, $r_h \in R$. The activity must use the resource for the entire processing time of the activity.

A solution to an ETSP problem is to assign start times to each activity such that:

1. the activities in a job are processed in the specified order
2. for each resource, the resource capacity constraints are respected
3. the total cost of the problem, c_J, is minimized.

For this paper, we further restrict our problem definition by placing the place the following restrictions on the ETSP model:

- There are m resources.
- Each of the m activities in a job requires a different resource.

These restrictions mean that the ETSPs addressed in this paper are extensions of the job shop scheduling problem which is known to be NP-hard [6].

We define the set of final activities from each job to be the *cost relevant activities*. These are the only activities that have a direct impact on the cost of a solution. In particular, if the start times of the cost relevant solutions are known (and participate in a feasible solution) then the cost of that solution is completely defined.

3 A Hybrid Local Search Approach

As the resources in the restricted version of the ETSP are unary, it is reasonable to examine and adapt local search techniques that have been applied to the job shop scheduling problem to ETSPs. In this section, we present the Hybrid Local Search (HLS) technique. We define the critical path calculation necessary to identify the activities that participate in the local search neighborhood, the moves that make up our neighborhood, a tabu search metaheuristic, and a linear model used to assign start times to a subset of the activities given a sequence.

3.1 Finding Critical Paths

An optimal solution to the job shop scheduling problem consists of a complete ordering of the activities on each resource such that the makespan is minimized. Local search in job shop problems, therefore, has focused on neighborhoods formed by identifying a critical path in a sub-optimal solution and swapping the order of a subset of the pairs of activities that are adjacent, on the critical path, and on the same resource. Such neighborhoods embedded in a tabu search have shown very strong performance [9, 16, 11]. The standard steps in creating a neighborhood for a job shop problem are:

1. to compute one or all of the critical paths in the current solution
2. to define a set of moves based on exchanging the positions of a subset of the adjacent activities on the identified critical path(s).

Given a (possibly sub-optimal) solution to a makespan minimization job shop scheduling problem, the critical path can be found by computing the longest path in a graph representing the solution [17, 4]. A special source node which precedes each activity is the starting point for the longest path algorithm and the sink node which follows each activity is the end point.

The definition and calculation of a critical path must be adapted when the makespan is not the minimization criteria. Kreipl [8] develops a large step random walk approach to solving the job shop scheduling problem with tardiness costs. In such a problem, the cost of a solution depends on how long after its due date each job finishes. Each job that finishes after its due date in a (possibly suboptimal) solution induces a critical path.

Kreipl calculates these critical paths by inserting a sink node for each job in the problem and computing the longest path from the source node to each of the sink nodes. This approach can be adapted for unary resource scheduling problems with both tardiness and earliness costs. Some modification is required however as it is necessary to take into account critical paths resulting from a tardy job as well as critical paths resulting from an early job.

Rather than following the approach of Kreipl, we choose to take advantage of the fact that we are using a constraint programming scheduling library as a basis on which to build our local search techniques. With such a library, constraint propagation maintains the minimum and maximum possible start times of each activity given the set of constraints and start time assignments that have been made. In particular, if we are given a sequence of activities on each resource and the optimal start time for each cost relevant activity, constraint propagation (arc consistency on the precedence constraints in the original jobs and in the sequence on each resource) removes inconsistent values for the start time domains of all other activities. Consider a job that is tardy, that is, the cost relevant activity, A, ends after the due date of the job. The convexity of the cost function means that if we could move A earlier in time, we could decrease the local cost. However, by definition of an optimal start time assignment, it is impossible to do this without increasing the cost from some other job at least as much as the local cost of A is reduced. In other words, there is a critical path of activities extending backward from A that prevents A from being moved earlier, either due to the cost considerations or do to feasibility. Furthermore, each of the activities in this critical path will have its start time domain reduced to a singleton by arc consistency. An analogous argument can be made to show that early jobs define a critical path that extends forward, preventing the activity from being moved later in time and so reducing local cost.

The result of this reasoning is the observation that given a sequence of activities on each resource and the optimal start time assignment for each cost relevant activity, arc consistency ensures that all activities that are in at least one critical path have singleton start time domains. Therefore, rather that computing a number of longest paths to find the activities that are on critical paths, we simply take the subset of activities that have singleton start time domains.

Two comments should be made here:

1. The arc consistency propagation on the precedence constraints implicitly computes the longest paths in the graphs that are necessary to identify the critical paths. We are not claiming to have created a way to identify such activities based on less information than is traditionally used. We are simply making use of the arc consistency techniques to reveal this information. It should be noted however that this is an example of the benefits of building scheduling algorithms on top of a constraint programming library. Not only did we not have to implement the longest path algorithms, we also, from an implementational efficiency perspective, benefit from the incrementality of the constraint propagation maintenance.

2. The condition of singleton start times domains is a necessary but not sufficient condition for membership in at least one critical path. In a one-resource, two-activity example, we could have the sole cost relevant activity, A, end at its due date, say, at time 10. If we assume that both activities have a duration of 5 time units and that A is sequenced after the other activity, B, then, B will have a

singleton start time domain [0..0] even though it is not part of any critical path. Therefore, the set of activities identified in this way is a superset of the activity critical activities.

3.2 A Simple Neighborhood for ETSPs with Unary Resources

A neighborhood commonly applied to job shop scheduling problems, and often denoted as *N1*, is formed by swapping all the pairs of activities that are adjacent, on the same resource, and on a critical path [9, 16]. While more complex neighborhoods, consisting of selecting a subset of *N1*, have been defined and shown to perform better (e.g., [11, 7]), it is unclear that the assumptions of the more complex neighborhoods carry directly over to ETSPs. For now, therefore, we choose the adapt the *N1* neighborhood to ETSPs.

This adaptation is straightforward. Given the set of critical activities identified as described above, we form our neighborhood by swapping (that is, exchanging the positions in the ordering) of all pairs of activities that are: in the set of critical activities, on the same resource, and adjacent.

More formally, given a solution, S, with a complete ordering of activities on each resource and optimal (given the activity orderings) start times assigned to the cost relevant activities, we define the set of moves involving the activities on one resource, R, as:

$$moves(S,R) = \left\{ swap(a,b) | a,b \in acts_R \wedge \left\{ \begin{array}{c} st_{min}(a) = st_{max}(a) \\ st_{min}(b) = st_{max}(b) \\ et_{max}(a) = st_{min}(b) \end{array} \right\} \right\} \quad (3)$$

Where:

- $swap(a,b)$ consists of reversing the order of a and b in the total ordering
- $acts_R$ are the set of activities executing on resource R
- $st_{min}(x)$ and $st_{max}(x)$ respectively indicate the minimum and maximum start time of activity x
- $et_{max}(s)$ is the maximum end time of activity x

The three conjunctions on the start and end times of the activities simply operationalize the criteria of belonging to the set of critical activities and being adjacent.

The set of all moves for S is the union of the moves for each resource.

3.3 A Tabu Search Metaheuristic

Given the above neighborhood, we use a simple tabu list that prevents the reverse swap for a randomly selected number of local search moves. For the experiments the number of moves for which an element is on the tabu list was randomly drawn with uniform probability from the interval [5, 14].

The only aspiration criterion is if a tabu move resulted in a solution that was better than any solution found so far. In such a case, the move is selected, if it is the current best possible move.

When we are at a search state such that all neighbors are either tabu or infeasible, we do the standard tabu list aging technique [11]: the oldest element is removed from the tabu list until a move can be selected.

3.4 Start Time Assignment using a Linear Program

The construction of the neighborhood relies on the ability to assign the optimal start times to the cost relevant activities given a complete ordering of the activities on each resource. Note that all we require is that the start times are optimal given the complete orderings; a different set of orderings may lead to a lower cost solution. To assign the start times we formulate a linear relaxation of the scheduling problem.[1]

For each activity a_{ji} we introduce a variable x_{ji} representing the start time of the activity. We also introduce a variable y_j, $j \in J$ that represents the cost associated with job j. The objective function is thus:

$$\text{minimize} \sum_{j \in J} y_j$$

and the constraints that define the problems are:

- All activities must start between their minimum and maximum start time

$$s_{min}(a_{ji}) \leq x_{ji} \leq s_{max}(a_{ji}) \quad \text{for } j \in J \text{ and } i \in \{1, \ldots, m\}$$

- Each activity of a job must start after the completion of the previous one

$$x_{ji} + pt(a_{ji}) \leq x_{ji+1} \quad \text{for } j \in J \text{ and } i \in \{1, \ldots, m-1\}$$

- A precedence relation between a pair of activities on the same resource, stating that a_{ji} is before a_{pq} is formulated

$$x_{ji} + pt(a_{ji}) \leq x_{pq}$$

- Each variable y_i is equal to the cost incurred if the corresponding job does not finish at its due date

$$y_i \geq tc_j(x_{jm} + pt(a_{jm})) - tc_j dd_j$$
$$y_i \geq ec_j dd_j - ec_j(x_{jm} + pt(a_{jm}))$$

  This relaxation is simply the one obtained by linearizing the piecewise linear expression (1) as described more generally in Refalo [15].

The linear relaxation we use does not take into account the resource capacity constraints. Since we have a convex cost function and since the problem matrix is integer and totally unimodular, using a simplex method for solving this model gives integer solutions.

When used as part of the local search technique, the set of start times found by the linear program solver represents a global solution because all activities on each resource are already sequenced. In the more general case that occurs in the CRS-Root and ProbePlus techniques described below, the linear program solution may violate some resource capacity constraints because not all activities on a resource are completely ordered.

[1] This linear formulation is an exact representation of the subproblem of assignment of optimal start times given an existing ordering of the activities on each resource. However, the same formulation is used as part of the CRS-Root and ProbePlus techniques presented below. As part of these techniques, the linear formulation is a relaxation of the global ETSP because a complete ordering of the activities on each resource may not exist. For completeness, therefore, we have chosen to present the linear model as a relaxation of the global ETSP.

4 Other Approaches

We compare our HLS approach to three constructive search techniques based on mixed integer programming, probing, and the identification and solving a cost relevant subproblem at the root node of a probe-based search. The final two techniques are also hybrid techniques, combining constraint and linear programming and are presented in more depth in Beck & Refalo [2].

4.1 A Mixed Integer Programming Technique

Solving single or multiple machine (or resource) scheduling problems is still a challenge for mixed-integer programming (MIP) techniques and several integer programming formulations have been proposed [14].

We experimented with two formulations. The first one is a time indexed formulation requiring a schedule horizon, T. For each activity i, it introduces T binary variables x_i^t, $t \in \{1, \ldots, T\}$. The variable x_i^t is set to 1 if the activity i starts at time t. Precedence constraints as well as resource capacity constraints can be modeled by linear constraints over these variables [14]. The major drawback of this approach is the size of the matrix which depends on T. For a 10×10 ETSP having a schedule horizon of 5000, this formulation needs approximately 500,000 binary variables and 50,000 constraints. With the horizon of problems considered, it was not possible to solve problems having more than 6 jobs and 6 resources with this approach with CPLEX 7.2 MIP optimizer.

The second formulation is a disjunctive formulation [1]. For each activity a_{ji}, we introduce a continuous variable S_{ji} representing its start time and we have the constraint:

$$s_{min}(a_{ji}) \leq S_{ji} \leq s_{max}(a_{ji})$$

A precedence constraint $a_{ji} \rightarrow a_{jk}$ is obviously modeled by the constraint:

$$S_{ji} + pt(a_{ji}) \leq S_{jk}$$

A resource constraint over a set of activities $\{a_{ui} \mid u \in R_i\}$ where R_i is the set of job indices processed by resource i is modeled by the constraints:

$$\left\{ \begin{array}{l} S_{pi} \geq S_{qi} + pt(a_{qi}) - K \times z_{pq} \\ S_{qi} \geq S_{pi} + pt(a_{pi}) - K \times (1 - z_{pq}) \\ z_{pq} \in \{0, 1\} \end{array} \right\} \text{for } p, q \in R_i, p < q$$

where K is some large constant that we have set to $T \times 10$ in our tests. The interpretation is that $z_{pq} = 1$ if a_{pi} is scheduled after a_{qi} and $z_{pq} = 0$ if a_{qi} is scheduled after a_{pi}.

We also introduce a variable y_j, $j \in J$ that represents the cost associated with job j. The objective function is thus:

$$\text{minimize} \sum_{j \in J} y_j$$

Each variable y_i is equal to the cost incurred if the corresponding job does not finish at its due date

$$\left\{ \begin{array}{l} y_i \geq tc_j(x_{jm} + pt(a_{jm})) - tc_j dd_j \\ y_i \geq ec_j dd_j - ec_j(x_{jm} + pt(a_{jm})) \end{array} \right\} for\ j \in J$$

This formulation is much smaller than the time-indexed one in terms of number constraints and variables. In this paper, we present experimental results using this formulation with CPLEX 7.2 MIP.

4.2 ProbePlus

The basic intuition behind the probe backtracking algorithm [5] is the use of probes in MIP solving. At a search state, a linear relaxation of the problem is solved. The relaxed optimal start times are used to generate resource profiles which represent the resource usage of the activities if they were assigned to their relaxed optimal start times. A peak, defined to be a point where the resource profiles exceeds the resource capacity, is then identified. If there is no such peak, then the relaxed optimal solution corresponds to a global solution. If such a peak exists, two activities contributing to the peak are heuristically selected and a branch is performed. One branch contains one sequence of the activity pair, the other branch contains the opposite sequence.

The *ProbePlus* technique adds one step to the basic probe backtracking approach. When the linear relaxation used in probe backtracking is solved, relaxed optimal start times are generated for each activity. The linear relaxation does not include the resource constraints and therefore the relaxed optimal solution is not necessarily a solution to the scheduling problem. In ProbePlus, we use the relaxed optimal start time of the cost relevant activities in a pure CP search to see if the relaxed optimal solution can be quickly extended to a true solution.

After solving the relaxed subproblem and before branching, the ProbePlus technique does a pure CP search as follows: first the relaxed optimal start times of the cost relevant activities are assigned, then a pure CP goal is used to try to extend this assignment to a global solution. If no such solution can be found, the search continues as in the Probe technique.

The ProbePlus technique obviously depends on the ease at which the relaxed start times of the cost relevant activity can be extended or shown to not correspond to any solution. The search however is likely to be in a highly constrained state because all the cost relevant activities have assigned start times. Our experiments show that standard constraint propagation techniques for scheduling (e.g., edge-finding [12]) almost always produce a solution or shown that none exists with minimal search. In a series of preliminary experiments, ProbePlus was found to be almost always better than probe backtracking.

Details of the scheduling heuristic and the creation of resource profiles can be found in Beck & Refalo [2]. The linear relaxation is the one described above in the context of HLS.

4.3 CRS-Root

As noted, in an ETSP only the final activity in each job has a direct impact on the optimization function. We make use of this fact to define a subproblem that only contains the final activities in each job. We refer to this subproblem as the *cost relevant subproblem* (CRS).

The CRS of an ETSP contains the resources and cost function of the full ETSP but only the cost relevant activities. A CRS is itself an instance of an ETSP with

each job containing one activity. Therefore, probe backtracking can be applied without modification to a CRS. A solution to a CRS provides two pieces of information that can be used in solving the full problem. First, it provides a lower bound on the cost of the full problem and, second, it provides a set of assignments for the cost relevant activities. If this set of assignments leads to a feasible solution to the full problem, we have found an optimal solution at the current search node.

In the CRS-Root technique, we generate a CRS using the resources, cost function, and cost relevant activities of the full problem. The time windows of the activities in the CRS are the corresponding time windows of those activities in the full problem. Two scheduling problems are then solved:

1. The CRS is solved, using probe backtracking.
2. The full model is solved using the start times for the cost relevant activities from the solution to the CRS. Because we already have assignments for the cost relevant activities, we search for a feasible solution using a simple, pure CP search strategy which ranks the activities on each resource.

If the second problem is successfully solved, we have found an optimal solution to the problem. In contrast, if no solution exists for the second problem, we can use the cost of the solution of the CRS as a lower bound on the optimal solution: all solutions must have a cost greater than or equal to the cost of the subproblem solution. However, as we have no solution to the full model, we must continue search.

If the CRS solution cannot be extended to a global solution, the rest of the search is equivalent to the ProbePlus search using the lower bound from the CRS solution.

The usefulness of this technique depends strongly on the effort required to solve the CRS and the subsequent effort required to solve the full problem using the start time assignments from the CRS solution. We can expect the effort required to solve the CRS to be much less than that of solving the full problem because of the much smaller number of activities in the CRS. We also expect that solving the full problem using the start time assignments from the CRS will be fast as argued above in the context of ProbePlus.

5 Empirical Studies

The goal of our experiments is to evaluate our hybrid local search technique (HLS) against the constructive techniques (MIP, ProbePlus, and CRS-Root) presented above.

All experiments were run on an 333 MHz, UltraSparc10, running SunOS 5.7, with 256M of RAM. For each problem instance, we imposed a maximum time limit of 1200 seconds. All algorithms are implemented using ILOG Scheduler 5.2, ILOG Hybrid 1.2, and ILOG CPLEX 7.2.

We conducted limited preliminary experiments with different parameterizations of the ILOG CPLEX MIP solver. The results reported here are those found with the best parameterization we were able to find: one in which the incumbent solutions are searched for more aggressively that with the default parameters. This is reasonable as our main interest is comparing the quality of sub-optimal solutions found by the various techniques. These parameter settings resulted in significantly better solutions in all problems where the optimality could not be proved and degraded performance in terms of proving optimality on only two problems (jb11 and ljb1 in Experiment 2).

5.1 Experiment 1: Random Problems

5.1.1 Experimental Problems

The first set of experiments use the problems originally defined in Beck & Refalo [2] which consist of nine sets of problems each with 10 problem instances.[2] Using the job shop problem generator of Watson et al. [20], we generated 10 job shop problems in each of three sizes: 10×10, 15×10, and 20×10. Each problem instance has a workflow structure and has the duration of each activity drawn with uniform probability from the interval [1, 99]. The workflow structure means that the resources in the problem are partitioned into two sets of approximately equal size. The activities in each job, then, must use each of the resources in the first partition before using any of the resources in the second. A randomly generated set of workflow job shop problems are more difficult that a similarly generated set of non-workflow job shop problems [19].

For each job shop problem instance, three earliness/tardiness problems are generated with different due dates and costs. The costs are uniformly drawn from the interval [1, 20]. The due dates are set by calculating, tlb, a lower bound on the job shop problem due to Taillard [16]. The due date for each job was then uniformly drawn from the interval:

$$[0.75 \times tlb \times lf, 1.25 \times tlb \times lf] \tag{4}$$

Where lf is the *looseness factor* and with possible values of {1.0, 1.3, 1.5}.

For each problem size and looseness factor, 10 earliness/tardiness cost problems are generated. This results in nine sets of problems each with 10 problem instances.

5.1.2 Results

In Table 1 we present a number of pieces of data. Each cell represents the results of runs on 30 problems, 10 of each size. The number of problems for which each technique found the best known solution (Best). For each problem instance, we identified the currently known lowest cost found by any of the techniques (including those in this paper). The table represents the number of problems (out of 30 problems in each cell) for which each technique found a solution with cost equal to the best known. The columns labeled "Unique" represent the number of problems for which a technique found a solution that is strictly better than the solutions found by all other techniques that have been applied to these problems. The best solution does not necessarily correspond to the optimal solution. In the columns labeled "Proven" we provide the number of problems for which each technique both found and proved the optimal solution within the time limit. Obviously this does not apply to incomplete techniques like HLS.

From Table 1 it can be seen that in terms of finding good solutions, HLS is better than ProbePlus across all levels of looseness. In comparing HLS to MIP, we see that MIP finds more best solutions for the tightest problems set (i.e., looseness of 1.0) while HLS performs better at looseness 1.3 and the performance is the same on the loosest set of problems. CRS-Root appears to be the best technique across all levels of looseness. It is interesting to note that on the hardest problems (those with looseness of 1.0) HLS finds 4 uniquely best solutions, therefore, at least on a few problems, it is performing strictly better than any of the techniques tested.

[2] These problems are available from the first author.

Looseness	1.0			1.3			1.5		
	Best	Unique	Proven	Best	Unique	Proven	Best	Unique	Proven
HLS	4	4	-	21	0	-	30	0	-
MIP	10	10	0	17	0	16	30	0	30
ProbePlus	7	1	0	9	0	7	21	0	13
CRS-Root	15	7	7	30	4	30	30	0	30

Table 1: The number of best, uniquely best, and optimal solutions found by each algorithm.

Size	10×10			15×10			20×10		
	Best	Unique	Proven	Best	Unique	Proven	Best	Unique	Proven
HLS	18	1	-	18	1	-	19	2	-
MIP	26	8	18	18	2	15	13	0	13
ProbePlus	12	0	10	13	0	5	12	1	5
CRS-Root	21	0	20	27	6	24	27	5	23

Table 2: The number of best, uniquely best, and optimal solutions found by each algorithm.

Table 2 presents the same data as Table 1 from the perspective of problem size. Each cell represents the results of 30 problems, 10 problems at each looseness factor.

Here again we see that in each problem class HLS outperforms ProbePlus. For the 10×10 problems the MIP technique is superior to HLS, while for the largest problems HLS dominates MIP. For the 15×10 problems performance of MIP and HLS is similar. In general, CRS-Root performs better than all other algorithms on the two larger problems, though for the 10×10 problems MIP finds more "best" solutions.

5.2 Experiment 2: Problems from the Genetic Algorithm Literature

In order to test the methods presented in this paper on a broader set of problems, we use a set of dynamic job shop scheduling problems from the Genetic Algorithm (GA) literature. These problems (originally due to Morton & Pentico [10]) have been used in a number of studies of GA scheduling techniques [18].

The deterministic, dynamic job shop scheduling problem, as defined by Vazquez & Whitley [18], is a job shop problem with release dates and due dates. A release date is the minimum time that any activity in a job can start execution. The due date is the time at which it is desired that the last activity in a job should end. A variety of optimization functions can be applied to these problems (e.g., weighted earliness, flow time, etc.). For this paper, we use the weighted earliness and tardiness which is defined as in the ETSPs (see Equations 1 and 2). The only difference with the dynamic JSP problems is that there is a single weighting for each job, meaning that the earliness and tardiness weights are equal. That is, in Equation 1, $ec_j = tc_j$ for each job j.

5.2.1 Results

It is standard in the GA literature to report the cost of the problem normalized by the weighted sum of the processing times of each activity. We follow this convention in reporting our results. The experimental details (hardware, software, time limit) are the same as used in Experiment 1.

Prob.	HLS	MIP	ProbePlus	CRS-Root	GA Best
jb1	0.241	0.191*	0.191*	0.191*	0.474
jb2	0.283	0.137*	0.137*	0.137*	0.499
jb4	0.568	0.568*	0.568*	0.568*	0.619
jb9	0.462	0.333*	0.862	0.862	0.369
jb11	0.374	0.213	0.704	0.213*	0.262
jb12	0.190	0.190*	0.190*	0.190*	0.246
ljb1	0.443	0.215	0.847	0.847	0.279
ljb2	0.754	0.603	1.190	1.364	0.598
ljb7	0.412	0.302	0.951	0.951	0.246
ljb9	2.301	1.765	2.571	2.571	0.739
ljb10	1.495	0.976	1.779	1.779	0.512
ljb12	1.404	0.835	1.601	1.601	0.399

Table 3: The best solution found by each algorithm for the GA problems.

Table 3 displays the best solutions found by each algorithm. The previous best results, in the "GA Best" column, are taken from Vazquez & Whitley [18]. To the best of our knowledge these represent the best known solutions to each problem. Problems where the optimal solution was found and proved are indicated by an asterisk. For the final five problems (ljb2-12), the optimal solutions remain unknown. For problem ljb2, the result found by MIP is the optimal solution, though this could not be proved within the time limit for this parameterization of the ILOG CPLEX MIP solver.[3]

The size of each problem and the CPU time taken by each algorithm to find, but not prove, the solution presented above are shown in Table 4.

Overall the MIP and CRS-Root technique performs best for the smaller problems while the existing GA approaches find better solutions for the larger problems. Of the non-GA techniques, MIP is clearly superior on the larger problems. Looking particularly at the performance of HLS, we can observe that it finds the optimal solution in two cases (jb4 and jb12). HLS finds equal or better solutions than ProbePlus for 10 problems, than MIP for 2 problems, and than CRS-Root for 9 problems.

The CPU times are slightly misleading as the solutions that were found are not the same. For example, for ljb12 the only solution found by ProbePlus and CRS-Root was found after about 0.8 seconds even though the total run time was 1200 seconds. In contrast, both HLS and MIP find better solutions in more time.

[3] As noted, in preliminary experiments the MIP technique using different hardware and parameters was able to prove the optimality of the solutions for jb11 and ljb1.

Problem	Size	HLS	MIP	ProbePlus	CRS-Root
jb1	10×3	1.20	0.13	0.3	0.32
jb2	10×3	3.34	1.01	48.91	28.58
jb4	10×5	1.79	0.11	0.44	0.4
jb9	15×3	31.75	28.55	281.85	184.36
jb11	15×5	31.49	583.43	593.75	381.85
jb12	15×5	31.85	1.74	740.62	192.84
ljb1	30×3	268.00	340.5	0.13	0.18
ljb2	30×3	162.87	344.73	1194.66	0.15
ljb7	50×5	1175.73	1127.85	0.54	0.52
ljb9	50×5	1179.83	125.14	0.86	0.85
ljb10	50×8	1183.05	796.68	0.83	0.82
ljb12	50×8	1159.03	149.35	0.8	0.82

Table 4: The size of each problem and CPU time in seconds for each algorithm.

6 Discussion

The HLS technique is competitive with existing techniques for solving ETSPs. However, in both experiments existing techniques perform better (CRS-Root in Experiment 1 and MIP in Experiment 2). Furthermore, as a incomplete technique, HLS cannot prove optimality. The other three techniques tested are complete.

There are a number of opportunities for improving HLS. First, no special effort was spent to create a good first solution. Indeed, the first solution used was simply found with a naive pure CP goal. For many of the problem instances (especially in Experiment 1) the local search was able to hill-climb all the way to the optimal solution, never using the tabu metaheuristic. We take this to indicate that the problems are relatively easy and that the initial solution was very poor. The latter point is supported by the fact that in many cases the first solution had a cost in the order of 10^5 or 10^6 while the optimal solution has a cost in the order of 10^2.

We can also look at adopting further techniques from the local search algorithms that have been applied to the job shop problem. In particular, we could consider smaller neighborhoods (which has been the trend for job shop [7]) as well as more sophisticated large step techniques [8] to use when the tabu search appears to be stuck in a suboptimal sub-space.

6.1 Hybrid Tree Search

The three important components of an efficient branch-and-bound search are: a branching strategy. lower-bounding techniques, and upper-bounding techniques (e.g., [3, 13]). In a previous work we have presented a branching strategy (based on probe backtracking), some lower bounding techniques such as solving a cost-relevant sub-problem and solving a linear relaxation, and some upper bounding techniques such as attempting to extend the solution to the cost-relevant subproblem to a global solution. In this paper we have presented and experimented with the HLS technique as a "stand-alone" local search approach to solving ETSPs. From the perspective of a branch-and-bound tree search, we

can also view HLS as an upper-bounding technique.[4] Therefore, aside from the direct improvements to the HLS technique noted above, we see an interesting area for future work in using HLS together with existing branch-and-bound components including the CPLEX MIP solver.

More broadly, we intend to investigate a more modular approach to hybrid tree search where various combinations of components from varying technologies can be used. For example, given the experimental results, it would be interesting to evaluate CRS+MIP+HLS: that is, a hybrid technique where the CRS is solved at the root node, the MIP branch-and-cut (and other standard MIP bounding techniques) is used for branching, and the HLS local search technique is used at the nodes of the MIP search to find upper-bounds.

7 Conclusion

In this paper, we presented a hybrid local search (HLS) procedure for solving earliness/tardiness scheduling problems. The HLS technique is based on a local search neighborhood defining a complete sequence of activities on each resource followed by the assignment of optimal start times (given the sequence) using a linear program solver.

Experimental results on two problems sets showed that the HLS technique is competitive with existing hybrid and MIP techniques for solving earliness/tardiness scheduling problems.

References

[1] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.

[2] J. C. Beck and P. Refalo. A hybrid approach to scheduling with earliness and tardiness costs. In *Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'01)*, pages 175–188, 2001.

[3] R.E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. MIP: Theory and practice – closing the gap. In M.J.D. Powell and S. Scholtes, editors, *Systems modelling and optimization: Methods, theory, and applications*, pages 19 – 49. Kluwer, 2000.

[4] M. Dell'Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231–252, 1993.

[5] H. El Sakkout and M. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *CONSTRAINTS*, 5(4):359–388, 2000.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[4] This is not an original observation. We believe, for example, that the Concorde TSP code (http://www.math.princeton.edu/tsp/concorde.html) includes local search techniques to find incumbent solutions.

[7] A.S. Jain, B. Rangaswamy, and S. Meeran. New and "stronger" job-shop neighborhoods: A focus on the method of nowicki and smitnicki (1996). *Journal of Heuristics*, 6(4):457–480, 2000.

[8] S. Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3), 2000.

[9] P. J. M. Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, January–February 1992.

[10] T.E. Morton and D.W. Pentico. *Heuristic Scheduling Systems*. John Wiley and Sons, 1993.

[11] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[12] W. P. M. Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.

[13] M.W. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33, 1991.

[14] M. Queyranne and A. Schulz. Polyhedral approaches to machine scheduling problems. Technical Report 408/1994, Departement of Mathematics, Technische Universitat Berlin, Germany, 1994. Revised 1996.

[15] P. Refalo. Tight cooperation and its application in piecewise linear optimization. In *Proceedings of Fifth International Conference on Principles and Practice of Constraint Programming (CP'99)*, pages 369–383, Alexandria, Virginia, October 1999. Springer-Verlag.

[16] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.

[17] E. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

[18] Manuel Vazquez and L. Darrell Whitley. A comparision of genetic algorithms for the dynamic job shop scheduling problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 1011–1018. Morgan Kaufmann, 2000.

[19] J-P Watson, J.C. Beck, A.E. Howe, and L.D. Whitley. Toward an understanding of local search cost in job-shop scheduling. In *Proceedings of the Sixth European Conference on Planning (ECP'01)*, 2001.

[20] J.P. Watson, L. Barbulescu, A.E. Howe, and L.D. Whitley. Algorithms performance and problem structure for flow-shop scheduling. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 688–695, 1999.