Fast Neighborhood Search For The Single Machine Earliness-Tardiness Scheduling Problem

Safia Kedad-Sidhoum[1] **and Francis Sourd**[2]
[1,2]Université Pierre et Marie Curie, Laboratoire LIP6, UMR7606
4, place Jussieu, 75 252 Paris Cedex 05, France
[1]e-mail: safia.kedad-sidhoum@lip6.fr
[2]e-mail: francis.sourd@lip6.fr

Abstract

This paper addresses the one machine scheduling problem in which n jobs have distinct due dates with earliness and tardiness costs. Fast neighborhoods are proposed for the problem. They are based on a block representation of the schedule and can be computed in $O(n^2)$. A timing operator is presented as well as swap and extract-and-reinsert neighborhoods. They are used in an iterated local search framework. Two types of perturbations are developed based respectively on random swaps and earliness and tardiness costs. Computational results show that very good solutions for instances with significantly more than 100 jobs can be derived in a few seconds.

Keywords: Machine scheduling, earliness-tardiness, neighborhood, iterated local search.

1 Introduction

We consider the one-machine scheduling problem where a set of n jobs $J_1, \ldots, J_n$ with processing times $p_1, \ldots, p_n$ has to be scheduled without preemption. For any i, J_i has a due date d_i. If the completion time of J_i, denoted by C_i, occurs after d_i, the job is late and a penalty $\beta_i(C_i - d_i)$ must be paid. Conversely, if $C_i < d_i$, the job J_i is early and the penalty is then $\alpha_i(d_i - C_i)$. The goal is to find a schedule π^* that minimizes the sum of all the penalties denoted by $\sum_{i=1}^{n} \max(\alpha_i(C_i - t), \beta_i(C_i - d_i))$.

This problem is known to be NP-hard. In the recent years, progress has been made in solving this problem. Sourd [15] propose a new branch-and-bound algorithm that improves the previous algorithm of Sourd and Kedad-Sidhoum [16]: it can solve instances with 50 jobs and more. In the special case where no idle time is permitted in between the tasks, Tanaka *et al.* [17, 18] propose algorithms that can solve even larger instances with up 200 tasks.

This problem appears in just-in-time environments for which meeting the requirements is still challenging. Minimizing the total weighted earliness and tardiness cost expresses the aim to reduce inventory cost and simultaneously satisfy the customer demands with timely delivery of products. This objective gives rise to non-regular performance measures, and thus leads to new issues

in designing scheduling algorithms. We can refer to the recent book of Józefowska [10] for a detailed survey on the models and algorithms developed in this area.

This paper re-addresses the heuristic approaches for the earliness-tardiness problem. Indeed, in view of the progress of the recent exact methods, the challenge is to derive, in a few seconds, very good solutions for instances with significantly more than 100 tasks. The improvement in computing very good lower bounds suggests to study heuristics based on the lower bounds. Bülbül *et al.* [1] study different ways to build good schedules from a preemptive relaxation of the problem. For the problem with time windows, Estève *et al.* [5] propose a recovering beam search that is guided by the lower bound of Sourd [13]. However, the computation of the lower bound is in general time consuming, which limits the size of the instances that are considered. Bülbül *et al.* [1] present experimental results for instances with up to 200 tasks but the heuristic takes several minutes. Similarly, Estève *et al.* [5] do not use the lower bound for their larger instances (that have between 100 and 130 tasks).

These remarks about the lower bound computation motivate the study of a fast local search scheme. Local search algorithms have been successfully applied to large instances. Among the competitive local search algorithms, James and Buchanan [8, 9] use a tabu search algorithm enhanced by a neighborhood scheme with a compressed solution space. We can also mention the work of Wan and Yen [20] and Tsai [19]. They have respectively developed a tabu search and a genetic algorithm. In all these approaches, solutions are typically represented as a *sequence* of tasks, that is the neighborhood consists of task moves or swaps in the sequence. A so-called *timing* algorithm is then called to compute the cost of each sequence, the algorithm computes the best schedule for the given sequence by optimally inserting idle time between tasks. Hendel and Sourd [7] give a list of the existing timing algorithm. The most efficient ones run in $O(n \log n)$ time. With such an approach, the best neighbor is found in $O(n^3 \log n)$ time, which is a bottleneck in an iterative improvement procedure. Indeed, the largest instances respectively tested by these authors contain 80 and 100 jobs. James and Buchanan [9] test their tabu search algorithm on instances with 250 jobs but running times are more than 1000s.

In order to illustrate the issue, let us consider a single local search (iterative improvement procedure) starting from a random sequence as the one described by Sourd and Kedad-Sidhoum [16]. The CPU time of one descent for a 50-job instance is about 0.05s but the time for a 100-job instance is more about 30s on a modern desktop. By contrast, the goal of this paper is to propose an efficient approach (that visits several local optima) running with a reasonable time limit of $n/20$s (5s are allowed for 100-job instances and 25s for 500-job instances).

To avoid the time spent exploring the neighborhood, Hendel and Sourd [6] propose an algorithm that combines the sequence moves and the timing procedure. The best sequence in a neighborhood is computed in $O(n^3)$ using a combined job interchanges and timing algorithm. They observe that computation time are indeed faster but the CPU time for 200-job instances is about 50s, which is too large for our purpose.

By contrast to these "*sequence-based*" approaches, we propose here an algorithm where each solution is represented by the real schedule, that is the task

start times and the blocks of tasks (a *block* is a maximal subsequence of tasks scheduled without idle time). We then propose a fast neighborhood search that can compute the best neighbor in $O(n^2)$ time while the size of the neighborhood is in $\Theta(n^2)$. The main features of this neighborhood is, first, to apply sequence moves inside each block (this algorithm is adapted from the work of Ergun and Orlin [4] for the single machine total weighted tardiness problem) and, second, to avoid the call to the timing procedure and replace it by a *timing operator* that moves each block backward or forward. We will see that this "*schedule-based*" neighborhood is not as large as the sequence-based neighborhood. However, when the number of blocks is significantly less than the number of jobs (which is usually the case for hard instances), this new neighborhood is good enough and leads to a drastic decrease in the computation times.

The second contribution of this paper is to embed our new neighborhood search in an iterated local search (ILS) in order to explore as many as possible local optima. We propose a new perturbation procedure that is very efficient for the search intensification. Experimental results show that, within the time limits above the resulting algorithm is able to find schedules that are less than 0.3% from the best known solution. For 50-job instances, for which the optimum is known, the deviation is less than 0.01%.

Section 2 presents the schedule-based neighborhoods and the algorithms to compute the best element in each neighborhood. Section 3 is devoted to the iterated local search and its different implementations. Section 4 describes detailed experimental results designed to set the efficiency of the method.

2 Schedule-based neighborhoods

As mentioned in the introduction, we do not consider feasible solutions as a sequence of tasks but as a detailed schedule, that is a vector $(C_1, \ldots, C_n)$ of the completion times of all the tasks. An alternative equivalent representation is based on the blocks of the schedule: for each block, the start time s is given with a subsequence of tasks that are adjacently scheduled after s.

Based on this representation, three neighborhoods will be defined:

- extract-and-reinsert in a block,
- swap in a block,
- subblock shift.

The first two neighborhoods modify the sequence of tasks while the last one changes the start times of the jobs of one block. We show that each of these neighborhoods can be explored in $O(n^2)$ time. The first two neighborhoods adapt the study of Ergun and Orlin [4] for the one-machine total weighted tardiness scheduling problem by taking into account earliness costs.

Let us introduce additional notations. For each job J_i, we will use the cost function $f_i(t) = \max(\alpha_i(d_i - t), \beta_i(t - d_i))$ that indicates the cost of J_i depending on its completion time. For instance, the cost of a schedule π, denoted by $c(\pi)$ is equal to $\sum_{i=1}^{n} f_i(C_i)$. The tasks of a schedule π are partitioned into blocks that are maximal sequences of tasks scheduled without idle time. Let $b(\pi)$ be the number of blocks. We will use the simpler notation b when the

reference to π is unambiguous. The blocks are denoted by $B_1(\pi), \ldots, B_b(\pi)$ or simply by $B_1, \ldots, B_b$. For some block $B = B_l$ of π, $l = 1, \ldots, b$, let $|B|$ denote the number of jobs of the block. $J_1^B, \ldots J_{|B|}^B$ denotes the jobs of B indexed in the order of schedule π. Similarly, we will use the notations f_i^B, p_i^B, C_i^B.

2.1 Extract-and-reinsert

For a schedule π, this neighborhood is obtained by removing a job J from its place in the schedule and reinserting it forward or back into the same block. The tasks between the initial and the new position of J are moved forward or backward so that J can be reinserted.

For a job J_i^B of a block B of π and a position $1 \leq j \leq |B|$ such that $i \neq j$, let v_{ij}^B denote the schedule derived from π by shifting J_i^B to position after j in B.

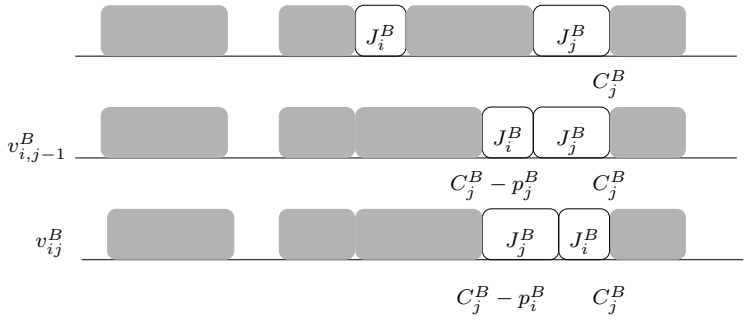

Figure 1: Extracting and reinserting J_i^B at positions $j-1$ and j

If we compare the neighbors v_{ij}^B et $v_{i,j-1}^B$ for $j < |B|$ and $i < j$, we can notice that they only differ in the positions of J_j^B and J_i^B: these two jobs are adjacently swapped (see Figure 1). Therefore, we have:

$$c(v_{ij}^B) - c(v_{i,j-1}^B) = f_i^B(C_j^B) + f_j^B(C_j^B - p_i^B) - f_i^B(C_j^B - p_j^B) - f_j^B(C_j^B)$$

which means that $c(v_{ij}^B)$ can be derived from $c(v_{i,j-1}^B)$ in $O(1)$ time.

We have $O(|B|)$ possible insertions for any job of B so the neighborhood associated to B can be explored in $O(|B|^2)$ time. Therefore, the neighborhoods of all the blocks are explored in $O(n^2)$ time.

2.2 Swap

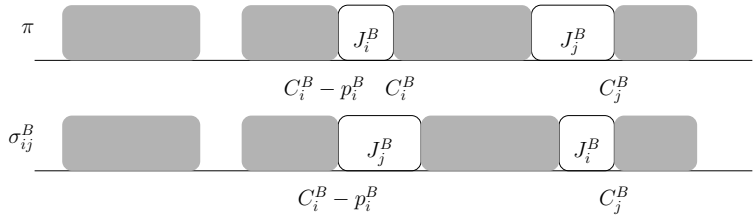

Figure 2: Swapping J_i^B and J_j^B

Given a schedule π, the swap neighborhood consists of swapping the positions of two jobs J_i^B and J_j^B of a block B ($1 \leq i < j \leq |B|$). Let σ_{ij}^B be the schedule derived from π after this swap.

The difficulty for computing the cost of σ_{ij}^B is that jobs between J_i^B and J_j^B move during the swap (see Figure 2). Therefore, we use the function $g_k^B(t)$ that gives the total earliness-tardiness cost for scheduling without idle time $J_k^B, \ldots, J_{|B|}^B$ starting at time t. With this definition, the total cost of σ_{ij}^B is given by:

$$\begin{aligned} c(\sigma_{ij}^B) = c(\pi) &+ f_i^B(C_j^B) - f_i^B(C_i^B) \\ &+ f_j^B(C_i^B - p_i^B + p_j^B) - f_j^B(C_j^B) \\ &+ g_{i+1}^B(C_i^B - p_i^B + p_j^B) - g_j^B(C_j^B - p_i^B) \end{aligned}$$

To ensure that the computation of the swap neighborhood is done in $O(|B|^2)$ time for block B, we are going to compute the values $g_{i+1}^B(C_{i-1}^B + p_j^B)$ and $g_j^B(C_j^B - p_i^B)$ for all i and j in $O(|B|^2)$ time.

The key point is that $g_i^B(t)$ is a piecewise linear convex function whose breakpoints can be determined in $O(|B|)$. More precisely, the breakpoints of g_i^B are given by the values of t for which $d_k^B = C_k^B - C_i^B + t$ for some $k \in \{i, \cdots, |B|\}$. Let $b_1, \cdots, b_r$ denote these breakpoints. The order of these breakpoints agrees with the non-increasing order of the value $C_k^B - d_k^B$ for $k \in \{i, \cdots, |B|\}$. In order to have all the jobs in the order of the breakpoints to which they are associated, we sort, in a preprocessing phase, the jobs of B in the non-increasing order of the values $C_k^B - d_k^B$.

Each function g_i^B is encoded by the sorted list of its breakpoints and its corresponding slopes. Each slope is a linear function in the interval $[b_k, b_{k+1}]$ for $k = 1, \ldots, r$. This sorted list is obtained in $O(|B|)$.

Let $\beta_i^B(t)$ (resp. $\alpha_i^B(t)$) be the weight of all jobs $j \in \{1, \cdots, i\}$ of B which are tardy (resp. early) when the starting time of B is t. The value of $g_i^B(0)$ is equivalent to determining all jobs that are early or late when $t = 0$ and computing $\alpha_i^B(0) + \beta_i^B(0)$. Then, the value of $g_i^B(t)$ for $0 < t \leq b_1$ is equal to $g_i^B(0) + t(\beta_i^B(0) - \alpha_i^B(0))$. Furthermore, let J_j^B be the job which becomes on time when $t = b_1$. Then, function $g_i^B(t)$ for $b_1 < t \leq b_2$ is equal to $g_i^B(t) = g_i^B(b_1) + (t - b_1)(\beta_i^B(b_1) - \alpha_i^B(b_1))$. We can notice that $\beta_i^B(b_1)$ and $\alpha_i^B(b_1)$ can be derived from $\beta_i^B(0)$ and $\alpha_i^B(0)$ in constant time. Indeed, when J_j becomes on time, then $\beta_i^B(b_1) = \beta_i^B(0) + \beta_j$ and $\alpha_i^B(b_1) = \alpha_i^B(0) - \alpha_j$. Hence, we can calculate $g_i^B(t)$ for all t such that $b_{k-1} < t \leq b_k$ in constant time given $g_i^B(b_{k-1})$, $\alpha_i^B(b_{k-1})$ and $\beta_i^B(b_{k-1})$.

We now consider how to efficiently evaluate the values $g_{i+1}^B(C_i^B - p_i^B + p_j^B)$ and $g_j^B(C_j^B - p_i^B)$ which are required to calculate $c(\sigma_{ij}^B)$. We then assume that, in the preprocessing step, the jobs of B have also been sorted in non-decreasing order of their processing time. This allows, for a fixed job J_i^B, to sort the values $C_i^B - p_i^B + p_j^B$ for all j in $O(|B|)$ time. Based on this order and on the order list of breakpoints of g_{i+1}^B, the values $g_{i+1}^B(C_i^B - p_i^B + p_j^B)$ (for all j) can be computed in $O(|B|)$ time. The same procedure can be used to compute $g_j^B(C_j^B - p_i^B)$ in $O(|B|)$ time for a fixed j and for all $i < j$. Therefore, the computation of the values $g_{i+1}^B(C_i^B - p_i^B + p_j^B)$ and $g_j^B(C_j^B - p_i^B)$ for all i and j is done in $O(|B|^2)$

time. Finally, the complete neighborhood of the schedule is explore in $O(n^2)$ time.

2.3 Subblock move

Let us consider a right subblock B of π, that is a sequence of adjacent tasks that is followed by an idle time. Its *local cost function*, given by $\delta_B(t) = \sum_{i \in B} f_i(C_i + t)$, represents the cost variation when the block is shifted by t. If the right derivative when $t = 0^+$ is negative, then the cost of the schedule can be decreased by shifting B to the right, that is by delaying it. The length of the shift corresponds to the maximum length such that the left derivative remains unchanged and the schedule remains feasible. This length can be determined in $O(|B|)$ time, where $|B|$ is the size of the subblock. Since there are at most n right subblocks, this neighborhood is also explored in $O(n^2)$ time.

A similar neighborhood is also defined for the left subblocks. It can also be explored in $O(n^2)$ time. It is important to notice that a local optimum for the neighborhood corresponds to an optimal timing. Indeed, it can not be improved using a timing algorithm.

After the execution of a subblock move, the block structure of the current schedule is changed and, therefore, it may be useful to try the other neighborhoods.

2.4 Preliminary experiments

We now study how these theoretical improvements behave in practice. We present in what follows a comparison of the implementations of the two families of neighborhoods. The first one, SeqNbh, is the "sequence-based" version while the second one, SchedNbh, is the "schedule-based" version. Both codes perform a local search starting with a random sequence and stopping as soon as a local optimum is found. SeqNbh is the piece of code used to find good initial solutions in the branch-and-bound of Sourd and Kedad-Sidhoum [16].

Both algorithms have been run on random instances of different sizes (the size is randomly fixed). Each instance is based on usual parameters of the earliness-tardiness scheduling literature. The processing times are generated from the uniform distribution $\{10, \cdots, 100\}$. The earliness and tardiness penalties are randomly chosen in $\{1, \cdots, 5\}$. There are two parameters to compute the due dates, namely, the *tardiness factor* τ and the *range factor* ρ. Both are randomly chosen in $[0.2, 0.8]$. Given τ, ρ and $\mathcal{J}$, the due date of each job J_j is generated from the uniform distribution $\{\theta_j, \cdots, \theta_j + \rho \sum_{k \in \mathcal{J}} p_k\}$ where $\theta_j = \max\{r_j + p_j, (\tau - \rho/2) \sum_{k \in \mathcal{J}} p_k\}$.

Figure 3 presents the result of these experiments. Each plot of the scatter graph corresponds to a run of either SeqNbh or SchedNbh on an instance. The x-coordinate is the size of the instance while the y-coordinate indicates the running time of the code. The graph clearly indicates two clusters of points which indicate that SchedNbh is significantly faster and can be used to solve instances with more than 100 jobs. By contrast, SeqNbh requires much CPU time as soon as there are more than 70 jobs.

In order to compare the quality of the local optima, we run SeqNbh and

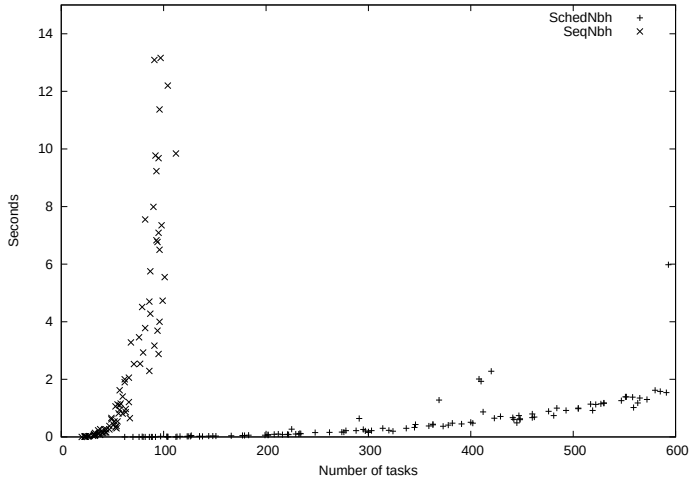

Figure 3: Comparison of the CPU times of SeqNbh and SchedNbh

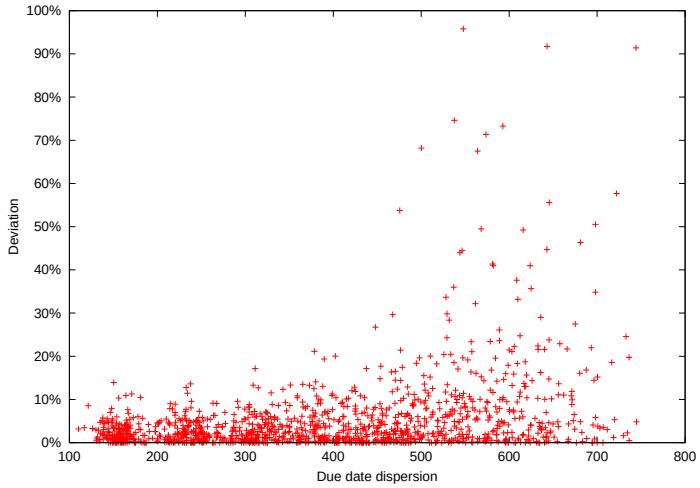

Figure 4: Deviation of SchedNbh with respect to the due date dispersion

SchedNbh on the 1274 50-job instances[1] that are exactly solved by Sourd [15]. The mean deviation from the optimum of SeqNbh is 1.65% and the maximal deviation is 27.2%. The mean deviation of SchedNbh is 5.34% and the maximal deviation is 95.8%. Unsurprisingly, since the sequence-based neighborhood is larger, SeqNbh is better. However, the performance of SchedNbh is quite good with respect to its running time, even if some very bad local optima may be met. Figure 4 displays the deviations of each of the 1274 local optima found by SchedNbh with respect to the statistical dispersion (variance) of the due dates. Instances with a larger dispersion means than idle time is more likely to be inserted between tasks and the schedule have more blocks. The figure shows that instances with larger dispersion have bad local optima, which can be explained by the fact that jobs are only moved inside a block: a schedule with more blocks has a smaller neighborhood.

3 Iterated local search

Preliminary tests of Section 2.4 motivates two axes to improve the efficiency of the local search. First, we can improve the quality of the neighborhood, which is studied in Section 3.1. Second, as computation times for a single descent are low, the search can be embedded in a more elaborated metaheuristic. Section 3.2 proposes three such schemes.

3.1 A composite neighborhood

To improve the quality of the neighborhood structure and escape from the bad local optimal, we implemented a neighborhood that composes one extract-and-reinsert move and one subblock move. This composite neighborhood can be explored in $O(|B|^3)$ time for each block B of the current schedule. Both in terms of computation times and quality, this neighborhood is halfway between SchedNbh and SeqNbH.

At this point, we can note that other composite neighborhood can be considered in order to derive larger neighborhood. Indeed, we can define a neighborhood that combines an extract-and-reinsert move and a swap or two swaps. As long as these moves are independent as defined by Congram *et al.* [2], a dynasearch neighborhood can be introduced. As shown by Ergun and Orlin [4], the dynasearch approach can be also used here as long as there is no block moves. In order to compose dynasearch moves and block moves, the pseudopolynomial algorithm of Sourd [14] can be used. However, in our case, its time complexity is too high to deal with large instances.

In order to limit the impact of the new neighborhood on the computation time, we first isolate the swap subneighborhood (which is also an extract-and-reinsert subneighborhood) that consists in swapping only adjacent jobs. This neighborhood contains only $O(n)$ schedules but it often contains an improving solution. It can be explored in $O(n)$ time and will be referred to as "adjacent swap"). Then, we hierarchize the neighborhood exploration in the following three levels:

1. Subblock move and adjacent swap

[1] Available at http://www-desir.lip6.fr/~sourd/project/et

2. Extract-and-reinsert and swap

3. Composite neighborhood

A neighborhood is (randomly) called only if all the neighborhoods at lower levels have been unsuccessfully explored. When a neighborhood at level 2 or 3 finds a new solution, the neighborhood search restarts at level 1.

To study the impact of this new neighborhood structure, we tested the variant of SchedNbh implementing the new neighborhood and the three-level hierarchy. For the 50-job benchmarks, the mean deviation is decreased from 5.34% to 4.05%. More significantly, the maximal deviation is decreased from 95.8% to 27.5%.

3.2 Iterated local search algorithms

Even with the improved composite neighborhood introduced in the previous section, a single local search descent may find a local optimum of poor quality. In order to find the best possible local optima, we embedded the local search descent into an iterated local search (ILS). This metaheuristic is very simple yet is usually very efficient for most problems [11]. In particular, ILS has successfully been used to tackle the single machine total weighted tardiness scheduling problem [2, 3].

The basic idea of the method is to iteratively run several local search descents. Each time a local optimum is reached, the starting point of the next local search is derived by applying a *perturbation procedure*. Typically, this procedure applies several non-improving moves the number of which is controlled by the so-called *strength parameter*. In this method, *intensification* is typically done during the local search phases while *diversification* is ensured by the perturbation phase.

3.2.1 Algorithm ILS

We first present a very simple implementation of the iterated local search. The perturbation phase consists in swapping s randomly selected pairs of tasks. Here, factor s corresponds to the strength of the perturbation. The two swapped tasks may belong to different blocks (it is important for a better diversification), which means that blocks durations are modified and then two blocks can collapse. In such a case, the collapsing blocks are merged into a single block.

3.2.2 Algorithm ILS2

We also propose a perturbation procedure that is not based on purely random swaps. This procedure is based on the information of the experimental results depicted in Figure 5. For a given arbitrary instance, six local search descents have been run starting with six different random solutions. For each descent, a point is obtained each time a new schedule is selected in the current neighborhood and a curve is built by linking all the points. The coordinates of each point are derived from the schedule by computing the sum of earliness costs and the sum of tardiness costs: these two values are respectively the x-coordinate and the y-coordinate.

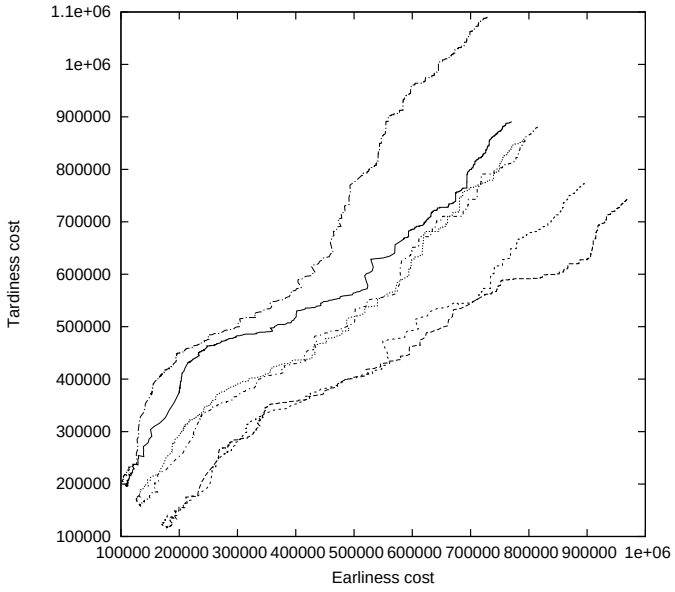

Figure 5: Six descents represented in a bicriteria space

This figure shows that the 6 curves are roughly parallel. We can then observe that the balance between earliness costs and tardiness costs does not change a lot during a single descent. Therefore, the aim of the perturbation procedure is to force the search to modify these ratios. In our implementation, the procedure first modifies the earliness/tardiness costs, that is, for each job, the new earliness penalty is randomly selected in $[(1 - s')\alpha_i, (1 + s')\alpha_i]$ and the new tardiness penalty is randomly selected in $[(1-s')\beta_i, (1+s')\beta_i]$. Second, with these modified costs, a local search descent is executed and the resulting local optimum becomes the pertubated schedule. The perturbation procedure completes by restoring original earliness-tardiness costs. In this procedure, s' indicates the strength of the perturbation. Clearly, we must have $0 \leq s' \leq 1$.

3.2.3 Algorithm ILS3

Algorithm ILS3 calls either the perturbation procedure of ILS with probability p or the perturbation procedure of ILS2 with probability $1-p$. The strengths s and s' of these two perturbation procedures are also parameters of ILS3. Another difference between ILS3 and the two previous algorithms is that the sequence of the initial schedule is not random but corresponds to the non-decreasing order of the due dates. Indeed, as it will be illustrated in Section 4, ILS and ILS2 are used to understand the behavior of the two perturbation procedures whereas the goal of ILS3 is to be the most efficient possible algorithm.

4 Experimental results

4.1 Instances

The first set of instances consists of the 1274 50-job instances of Sourd and Kedad-Sidhoum [16] whose optimal values are all known [15]. These instances

ILS

n	$s=1$	$s=2$	$s=4$	$s=8$	$s=16$	$s=32$
50	0.01%	0.01%	0.01%	0.02%	0.02%	0.04%
100	0.08%	0.07%	0.07%	0.10%	0.14%	0.20%
200	0.28%	0.27%	0.28%	0.33%	0.38%	0.47%
300	0.34%	0.36%	0.31%	0.35%	0.41%	0.50%
400	0.42%	0.41%	0.38%	0.40%	0.44%	0.48%
500	0.46%	0.45%	0.44%	0.41%	0.47%	0.50%

ILS2

n	$s'=0.15$	$s'=0.3$	$s'=0.45$	$s'=0.6$	$s'=0.75$	$s'=0.9$
50	2.25%	0.69%	0.20%	0.04%	0.03%	0.01%
100	2.33%	0.63%	0.13%	0.07%	0.08%	0.12%
200	1.49%	0.54%	0.27%	0.27%	0.36%	0.44%
300	1.03%	0.57%	0.39%	0.43%	0.46%	0.56%
400	0.76%	0.58%	0.46%	0.54%	0.56%	0.64%
500	0.65%	0.47%	0.47%	0.61%	0.64%	0.69%

Table 1: Mean deviation for ILS and ILS2 with different strength values

are clearly useful to prove the efficiency of the Iterated Local Search approach.

As the goal of the paper is to tackle with larger instances, we generated new instances with 100, 200, 300, 400 and 500 tasks according to the same generation scheme. The processing times are generated from the uniform distribution $\{10, \cdots, 100\}$. The earliness and tardiness penalties are randomly chosen in $\{1, \cdots, 5\}$. There are two parameters to compute the due dates, namely, the *tardiness factor* τ and the *range factor* ρ. Given τ, ρ and $\mathcal{J}$, the due date of each job J_j is generated from the uniform distribution $\{\theta_j, \cdots, \theta_j + \rho \sum_{k \in \mathcal{J}} p_k\}$ where $\theta_j = \max\{r_j + p_j, (\tau - \rho/2) \sum_{k \in \mathcal{J}} p_k\}$.

For each $n \in \{100, 200, 300, 400, 500\}$, $\tau \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$ and $\rho \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$, 10 instances are randomly generated, that is we have 2450 instances with 100 jobs or more.

4.2 Results for ILS and ILS2

Each of algorithm ILS and ILS2 has one parameter, namely s and s' respectively. In this section, we compare these algorithms for different values of s and s'. Value s varies in $\{1, 2, 4, 8, 16, 32\}$ and s' varies in $\{0.15, 0.3, 0.45, 0.6, 0.75, 0.9\}$. We will use notations ILS(s) and ILS2(s') when the reference to the value of the strength is necessary.

We first test ILS and ILS2 on the 1274 50-job instances, whose optima are all known [15]. Therefore, we are able to compute the deviation of the two ILS algorithms. For a time limit of only 0.5 second, the mean deviation of ILS (with $s = 4$) is equal to 0.01%, which is very good. For the same instances, the deviation of Heur50 described by Sourd and Kedad-Sidhoum [16] is 0.1% with larger computation times. Moreover, the optimum is found for 93% of the instances and the maximal deviation is 1.47%. For the other values of s, results are very similar, as indicated in the first line of Table 1.

	ILS					
n	$s=1$	$s=2$	$s=4$	$s=8$	$s=16$	$s=32$
100	40.20%	37.96%	36.33%	29.18%	22.65%	15.10%
200	5.92%	5.71%	3.88%	1.84%	1.02%	0.61%
300	3.88%	3.88%	3.88%	3.06%	1.02%	0.41%
400	3.47%	4.29%	3.88%	3.06%	2.04%	0.20%
500	5.51%	3.88%	3.88%	3.47%	3.06%	1.02%

	ILS2					
n	$s'=0.15$	$s'=0.3$	$s'=0.45$	$s'=0.6$	$s'=0.75$	$s'=0.9$
100	5.10%	39.18%	53.88%	49.80%	36.73%	24.90%
200	4.08%	19.39%	10.82%	6.33%	1.63%	1.22%
300	7.76%	15.31%	7.76%	2.24%	1.84%	0.00%
400	17.35%	12.04%	4.69%	2.24%	0.82%	0.00%
500	15.51%	16.33%	6.12%	0.82%	0.20%	0.20%

Table 2: Best known solutions found by ILS and ILS2

Therefore, for small instances, ILS is nearly optimally whatever the strength s is. The efficiency of ILS2 is comparable only for $s' \geq 0.6$. For small values of s', we can then conclude that there is no enough diversification because many returned scheduled are more than 20% from the optimum.

For instances with 100 tasks or more, we do not know the optimum. Instead, we are going to use the best known upper bound, which is in fact the best solution found in all our experimental campaign. We tried to solve each instance with at least 30 different variants of ILS, ILS2 or ILS3 (not all the experiments are reported here). Table 1 shows the average deviations of the six variants of ILS and the six variants of ILS2. We recall that the computation times are fixed to $n/20$ seconds.

In most cases, the mean deviation is less than 1%. The only exception is for ILS2 with $s' = 0.15$, where we can conclude that the search is not diversified enough. This weak deviation is remarkable since all these local search procedures start with random schedules. Therefore, these solutions are very likely near optimal. When comparing the efficiency of ILS and ILS2, the results of Table 1 tend to prove that ILS is better than ILS2. However, this conclusion is counterbalanced, by Table 2. This table indicates the number of times that ILS(s) or ILS2(s') finds the best known upper bound (actually the percentage over the 490 instances of each size). Here it appears that ILS2(0.3) is the algorithms that is the most efficient at finding the best solution. The explanation is that the perturbation procedure of ILS2 is good at finding promising schedules that are not too different from the current solution: therefore it is useful to explore a medium size region. Conversely, the perturbation procedure of ILS helps to explore larger regions.

4.3 ILS3

The above remarks motivate algorithm ILS3 that combines ILS(8) and ILS2(0.3). After some preliminary tests, the value $p = 0.25$ is chosen. It means there

	$n = 100$	$n = 200$	$n = 300$	$n = 400$	$n = 500$
Mean deviation	0.05%	0.19%	0.21%	0.29%	0.34%
Max deviation	1.01%	1.54%	1.96%	3.96%	3.35%
Best found	51.0%	11.2%	14.7%	12.9%	8.9%
Less than 0.5%	98.6%	92.4%	88.6%	83.1%	79.0%
Less than 1%	99.8%	98.2%	97.4%	95.5%	93.4%

Table 3: Performances of ILS3

are three ILS2 "small" perturbations for one ILS "large" perturbation. Table 3 presents some statistics on the behavior of ILS3. The mean deviation is smaller than any tested variant of ILS and ILS2. Even if ILS3 does not find the best known solution as often as ILS2(0.3), it achieves a good compromise between the difference performance criteria. The maximal deviation is for instance less than 4% while it is typically between 5% and 15% for ILS and ILS2.

In order to have a more complete view of the deviation, Figure 6 compares the cumulative distribution curves of the deviations. For a given deviation x in abscissa, the corresponding y-coordinate indicates the percentage of instances for which the deviation is less than x%. The curve is generated from the class of instances with 400 tasks. The figure confirms that ILS2 finds more solutions with low deviations than ILS but ILS has a larger proportion of solutions with higher deviation. Interestingly, ILS3 combines the advantage of both methods since it is comparable to ILS2 for small deviation and to ILS for large deviations.

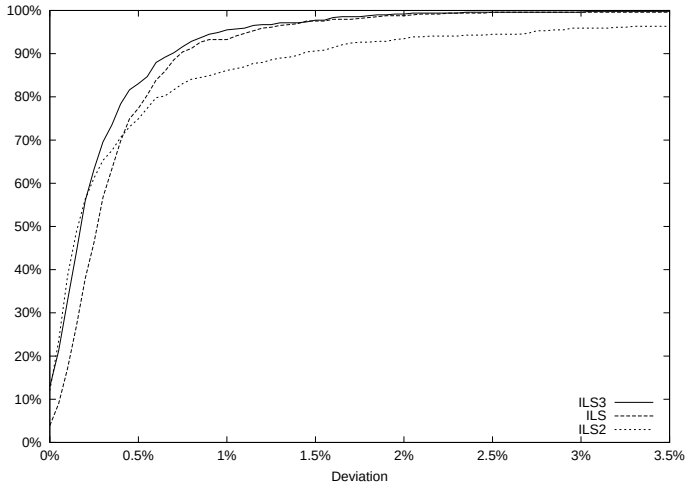

Figure 6: Cumulative distribution for different deviations

Figure 7 shows the same cumulative distribution curves of the deviations for the different size of instances. It shows that the performance slightly decreases with the size of the instances. It is not surprising since the increase in the time limit is linear while the complexity of the neighborhood search is quadratic — not to mention that the number of steps in the local search is pseudo-polynomial and the search space is factorial.

For 50-job instances, for which the optimum is known, the deviation is less than 0.01%. In order to evaluate the quality of the ILS3 solutions for large

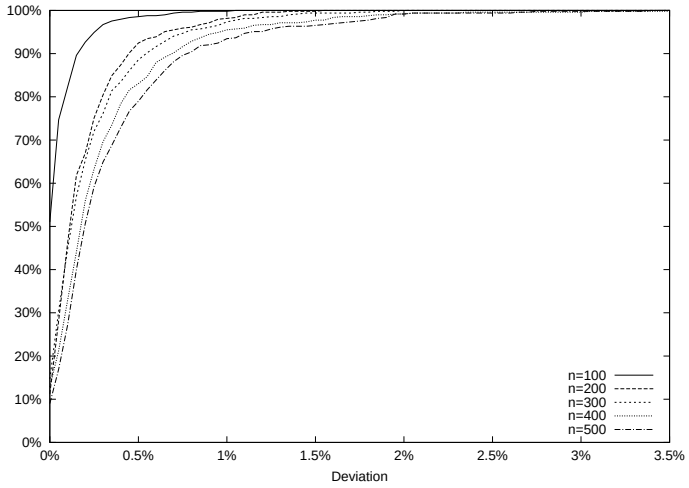

Figure 7: Performance of ILS3 for different sizes of instances

	$n = 100$	$n = 200$	$n = 300$	$n = 400$	$n = 500$
Mean deviation	5.59%	3.75%	2.90%	5.33%	5.53%
Max deviation	16.44%	16.21%	23.75%	25.88%	18.96%

Table 4: Deviations from LB

instances, we compute Lagrangean based lower bounds described by Kedad *et al.* [12]. We can mention that the computation of the reinforced Lagrangean lower bound described by Sourd [15] is time-consuming. Table 4 shows the average and maximum deviation between the lower bound denoted by LB and the best solution found by ILS3 for instances of size 100, 200, 300, 400 and 500. Each value is the percentage over the 490 instances of each size. For the 400-job and 500-job instances, the Lagrangean optimum is not always reached due to CPU time restrictions. Regarding these results, we can conclude that the worst mean deviation is around 5.5 which means that the solutions obtained by ILS3 are close to the optimal values.

The last series of experiments is devoted to the question of how ILS3 behaves when more CPU time is allowed. For all the 200-job instances, the solutions found after 5s, 10s, 20s and 40s are recorded. Figure 8 shows the distribution of the deviations of the solutions found at each of these time limits. It shows that the performance of ILS3 regularly improves with larger computation times. After 5s, the average deviation is 0.33% and the maximal deviation is 1.91%. After 40s, the average deviation is only 0.06% and the maximal deviation is 0.95%. These results show that ILS3 has a good balance between intensification and diversification.

5 Concluding remarks

This paper has addressed the one machine earliness-tardiness scheduling problem with a focus on heuristic approaches. Iterated local search algorithms based on fast neighborhoods have been developed. They rely on a new block representation of a solution. The diversification of the search is based on both random

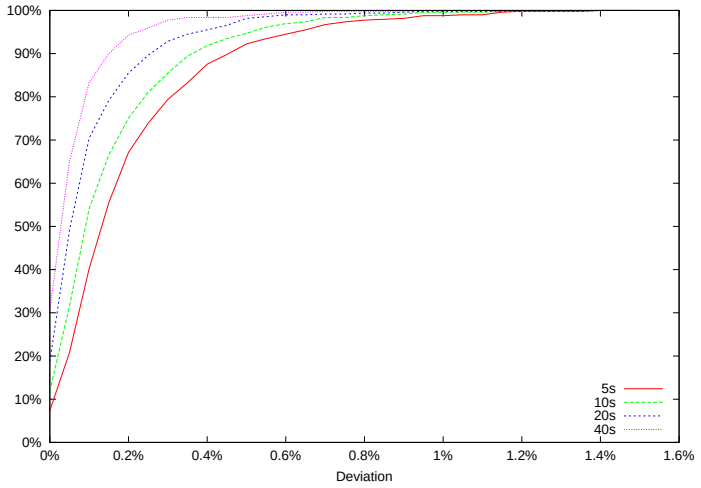

Figure 8: Performance of ILS3 for different time limits

swaps and earliness and tardiness costs perturbations. Experimental tests show that very good solutions for instances with significantly more than 100 jobs can be derived in a few seconds. For 50-job instances, for which the optimum is known, the deviation is less than 0.01% with very low computation times. Further work could consist in enhancing the size on the fast neighborhoods by efficiently allowing moves between blocks. An extensive study should be driven to decompose the neighborhood into a more elaborated hierarchy of subneighborhoods. Learning algorithms could also be used to automatically guide the search through promising neighborhoods. Other natural issues are to extend the approach to solve more general problems. Indeed, the schedule-based representation of a solution offers a better framework to tackle industrial constraints.

References

[1] K. Bülbül, P. Kaminsky, and C. Yano, *Preemption in single machine earliness/tardiness scheduling*, Journal of Scheduling **10** (2007), 271–292.

[2] R.K. Congram, C.N. Potts, and S.L. van de Velde, *An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem*, INFORMS Journal on Computing **14** (2002), 52–67.

[3] M. den Besten, T. Stützle, and M. Dorigo, *Design of iterated local search algorithms*, Proceedings of the EvoWorkshops on Applications of Evolutionary Computing, vol. 2037, 2001, pp. 441–451.

[4] Ö. Ergun and J.B. Orlin, *Fast neighborhood search for the single machine total weighted tardiness problem*, Operations Research Letters **34** (2006), 41–45.

[5] B. Estève, C. Aubijoux, A. Chartier, and V. Tkindt, *A recovering beam search algorithm for the single machine just-in-time scheduling problem*, European Journal of Operational Research **172** (2006), 798–813.

[6] Y. Hendel and F. Sourd, *Efficient neighborhood search for the one-machine earliness-tardiness scheduling problem*, European Journal of Operational Research **173** (2006), 108–119.

[7] ———, *An improved earliness-tardiness timing algorithm*, Computers & Operations Research **34** (2007), 2931–2938.

[8] R.J.W. James and J.T. Buchanan, *A neighbourhood scheme with a compressed solution space for the early/tardy scheduling problem*, European Journal of Operational Research **102** (1997), 513–527.

[9] ———, *Performance enhancements to tabu search for the early/tardy scheduling problem*, European Journal of Operational Research **106** (1998), 254–265.

[10] J̈. Józefowska, *Just-in-time scheduling*, Springer-Verlag, 2007.

[11] H.R. Lourenço, O. Martin, and T. Stützle, *Iterated local search*, Handbook of Metaheuristics, Kluwer Academic Publishers, 2006, Volume 57 of International Series in Operations Research & Management, pp. 321–353.

[12] Kedad-Sidhoum S., Y. A. Rios Solis, and F. Sourd, *Lower bound for the earliness-tardiness scheduling problem on parallel machines with distinct due dates*, European Journal of Operational Research **3** (2008), 1305–1316.

[13] F. Sourd, *The continuous assignment problem and its application to preemptive and non-preemptive scheduling with irregular cost functions*, INFORMS Journal on Computing **16** (2004), 198–208.

[14] ———, *Dynasearch neighborhood for the earliness-tardiness scheduling problem with release dates and setup constraints*, Operations Research Letters **34** (2006), 591–598.

[15] ———, *A reinforced Lagrangean relaxation for non-preemptive single machine problem*, INFORMS Journal on Computing (2008), to appear.

[16] F. Sourd and S. Kedad-Sidhoum, *A faster branch-and-bound algorithm for the earliness-tardiness scheduling problem*, Journal of Scheduling **11** (2008), 49–58.

[17] S. Tanaka, *An exact algorithm for single-machine scheduling without idle time*, 3rd Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA), 2007, pp. 314–317.

[18] S. Tanaka, S. Fujikuma, and M. Araki, *A branch-and-bound algorithm based on lagrangian relaxation for single-machine scheduling*, International Symposium on Scheduling, 2006, pp. 148–153.

[19] T.I. Tsai, *A genetic algorithm for solving the single machine earliness/tardiness problem with distinct due dates and ready times*, International Journal of Advanced Manufacturing Technology **31** (2007), 994–1000.

[20] G. Wan and B.P.C. Yen, *Tabu search for single machine with distinct due windows and weighted earliness/tardiness penalties*, European Journal of Operational Research **142** (2002), 271–281.