A Fast Algorithm for Searching Insertion, Swap, and Twist Neighborhoods for the Single Machine Total Weighted Tardiness Problem

Özlem Ergun * James B. Orlin †

22 March 2004

Abstract

Most successful heuristics for solving \(1|\sum w_j T_j\) are based on swap moves. We present an algorithm which improves the complexity of searching the swap neighborhood from \(O(n^3)\) to \(O(n^2)\). We show that this result also improves the complexity of the dynasearch heuristics by Congram et al. [1] and Grosso et al. [3].

Key words: neighborhood search, single machine scheduling, algorithms

1 Introduction

Given a set of jobs \(J = \{1, 2, ..., n\}\) with processing time \(p_j\), weight \(w_j\), and due date \(d_j\) for every job \(j \in J\), the problem we consider seeks to find a schedule of the jobs on a single machine that minimizes the total weighted tardiness. Let \(C_j\) be the completion time of job \(j\). The tardiness of job \(j\), \(T_j\), is defined as \(\max\{C_j - d_j, 0\}\). We will use the classic \(1|\sum w_j T_j\) notation to denote the single machine total weighted tardiness problem.

Lawler [4] and Lenstra et al. [5] show that \(1|\sum w_j T_j\) is strongly NP-hard. Among exact solution approaches the branch and bound algorithm by Potts and Van Wassenhove [7] solve problems up to 40 jobs. On the other hand, heuristic algorithms have been successfully applied to larger instances with up to 100 jobs. Most local search algorithms used are based on insertion and swap moves.

Among the competitive heuristics Congram et al.[1] use an exponential sized dynasearch swap neighborhood which is searched in polynomial time. The dynasearch neighborhood, introduced in [6], consists of combining non-overlapping single moves. In [1] the single moves are swaps. Grosso et al. [3] enhance the dynasearch swap neighborhood by adding generalized insertion moves.

*Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332. E-mail: oer-gun@isye.gatech.edu. Özlem Ergun was supported in part under NSF grant DMI-0238815.

†Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139. E-mail: jorlin@mit.edu. James Orlin was supported in part under NSF grant DMI-0217123.
With a straightforward algorithm it takes $O(n^3)$ time to search the swap neighborhood for $1||\sum w_j T_j$. In [1] and [3] it is shown that the dynasearch swap neighborhood and its enhanced version are also searched in $O(n^3)$ time via dynamic programs. Both papers improve the average computational times by implementing elimination based speed ups. In [2] a special balanced tree data structure was used to improve the search time of the swap neighborhood and its dynasearch extension to $O(n^2 \log n)$.

In this paper, we present efficient algorithms which search the swap and twist neighborhoods and their dynasearch versions for $1||\sum w_j T_j$ in $O(n^2)$ time. Furthermore, we show that the dynasearch neighborhood based on the combination of insertion, swap, and twist neighborhoods is also searched with the same time complexity. This combined neighborhood includes the dynasearch neighborhood of [3].

2 Neighborhoods

Given a set of jobs $J = \{1, \ldots, n\}$, we denote a schedule by a permutation $\pi$ of $\{1, \ldots, n\}$, where $\pi(i)$ denotes the job that is scheduled in the $i$th position. Let $\pi^*$ be the permutation $(1, \ldots, n)$. A neighborhood $N$ of a schedule is a subset of permutations. The neighborhood of schedule $\pi$ is $N(\pi) = \{\pi \circ \sigma : \sigma \in N\}$, where $\circ$ denotes functional composition, that is, $\pi \circ \sigma(j) = \pi(\sigma(j))$. In the rest of the paper we assume without loss of generality that the current schedule is $\pi^*$ and thus its neighborhood is $N(\pi^*) = N$.

For a real number $x$, let $x^+ = \max(x, 0)$. Let $P_j = \sum_{i=1}^j p_i$. We let $T_i = (P_i - d_i)^+$ denote the tardiness of job $i$ in $\pi^*$. For any schedule $\pi$, we let $c(\pi)$ denote the objective function for the schedule, for example, $c(\pi^*) = \sum_{i=1}^n w_i (P_i - d_i)^+$.

In the following subsections, we discuss three neighborhoods for the single machine weighted sum of tardiness scheduling problem. We present algorithms for searching each one of these neighborhoods and their dynasearch versions in $O(n^2)$.

2.1 The Insertion Neighborhood

In this subsection, we consider the neighborhood consisting of insertions. Given a schedule $\pi$, its insertion neighborhood consists of all schedules that can be obtained by removing a job from its place in $\pi$ and re-scheduling it. Let $v_{ij}$, for $i \leq j$, be the permutation obtained from $\pi^*$ by moving job $i$ to the position after job $j$. That is

$$v_{ij}(k) = \begin{cases} 
  k & \text{if } k < i \text{ or } k > j \\
  k + 1 & \text{if } i < k \leq j \\
  j & \text{if } k = i.
\end{cases}$$

Note permutation $\tilde{v}_{ij}$, for $i \geq j$, can be defined in a similar way as the permutation obtained from $\pi^*$ by moving job $i$ to the position before job $j$. The discussion for $v_{ij}$ directly applies to $\tilde{v}_{ij}$ with minor modifications.
Using this notation, we observe the following:

\[
\begin{align*}
c(v_{ii}) &= c(\pi^*) \text{ for all } i; \\
c(v_{ij}) - c(v_{i,j-1}) &= w_j(P_j - p_j - d_j)^+ - w_j(P_j - d_j)^+ + w_i(P_j - d_i)^+ - w_i(P_{i-1} - d_i)^+ \text{ for all } i < j.
\end{align*}
\] (1)

**Theorem 1** The time to compute \(c(v_{ij})\) for all \(i \leq j\) is \(O(n^2)\).

**Proof.** For a fixed \(i\), from \(c(v_{i,j-1})\), \(c(v_{ij})\) is calculated in \(O(1)\) time. Hence \(O(n)\) time is required to evaluate \(c(v_{ij})\) for all \(j\). See [3] for an alternative proof. ■

### 2.2 The Swap Neighborhood

In this subsection, we consider the neighborhood that consists of swapping the positions of two jobs \(i\) and \(j\) for all \(1 \leq i < j \leq n\). We let \(\sigma_{ij}\) be the permutation that is obtained from \(\pi^*\) by swapping the elements in positions \(i\) and \(j\). That is

\[
\sigma_{ij}(k) = \begin{cases} 
  k & \text{if } k \neq i \text{ or } j \\
  j & \text{if } k = i \\
  i & \text{if } k = j.
\end{cases}
\]

Next we establish the main result of the paper which improves the search time of the swap neighborhood to \(O(n^2)\) from the previous \(O(n^3)\) bound. (See for example [1] and [3].)

First, we describe preprocessing steps and the lists these steps create. We sort jobs in order of increasing processing times. Let \(\text{ProcessingList}\) be a list of jobs 1 to \(n\) sorted in non-decreasing order of processing times. Let \(\text{LateList}\) be an ordered set of jobs 1 to \(n\) sorted in non-increasing order of lateness with respect to the current schedule \(\pi^*\). One can obtain \(\text{ProcessingList}\) and \(\text{LateList}\) in \(O(n \log n)\) time using standard sorting techniques.

Let \(W(i)\) be the total weighted tardiness due to the first \(i\) jobs. Thus

\[
W(i) = \sum_{k=1}^{i} w_k(P_k - d_k)^+.
\]

We can compute \(W(i)\) for all \(i\) in \(O(n)\) time.

Let \(g_k(t)\) denote the weighted sum of tardiness if we schedule only jobs \(k, k+1, \ldots, n\) and in that order, and if job \(k\) starts at time \(t\). Thus

\[
g_k(t) = \sum_{i=k}^{n} w_i(P_i - P_{i-1} + t - d_i)^+.
\]

Using this notation, we observe the following:

\[
c(\sigma_{ij}) = [W(i - 1)] + \sum_{k=i}^{j} \left[w_k(P_k - d_k)^+ + w_j(P_{j-1} + p_j - d_j)^+ + \left[g_{i+1}(P_{i-1} + p_j) - g_j(P_j - p_i)\right] + [W(n) - W(j)]\right] \text{ for all } i \leq j.
\] (3)
The first term of the right hand side of (3) is the weighted tardiness of the first \( i-1 \) jobs. The second and third terms give the weighted tardiness of jobs \( i \) and \( j \) after they are swapped. The fourth term gives the weighted tardiness of jobs \( i+1 \) to \( j-1 \) after the swap. The final term accounts for the weighted tardiness of jobs \( j+1, j+2, \ldots, n \).

Hence to compute \( c(\sigma_{ij}) \) for all \( i \) and \( j \) in \( O(n^2) \) time, it suffices to compute \( g_{i+1}(P_{i-1}+p_j) \) and \( g_j(P_j-p_i) \) for all \( i \) and \( j \) in \( O(n^2) \) steps.

We next show how to compute \( g_{i+1}(P_{i-1}+p_j) \) for a given \( i \) and for all \( j > i \) in \( O(n) \) steps. We note that \( g_{i+1}(t) \) is a piecewise linear convex function of \( t \). Furthermore, its slope changes only at values of \( t \) for which \( d_k = P_k - P_i + t \) for some \( k \in \{i+1, \ldots, n\} \). We refer to these values as the breakpoints.

The order of the breakpoints agrees with the order of the jobs on \( LateList \), and so one can determine the breakpoints of \( g_{i+1} \), in order, in \( O(n) \) time. Let \( b_1, \ldots, b_r \) denote these breakpoints. Then one can express the function \( g_{i+1} \) as a set of linear functions in the intervals \([b_k, b_{k+1}]\) for \( k = 1 \) to \( r - 1 \). To determine all of these intervals and the linear functions on the intervals takes \( O(n) \) time.

For example, determining the value of \( g_{i+1}(0) \) is equivalent to determining all jobs that are late when \( t = 0 \) and their contribution to the weighted tardiness. Let the weight of all jobs \( j \in \{1, \ldots, i+1\} \) which are late when scheduled starting at time \( t \) be \( w_{i+1}(t) \). Then the value of \( g_{i+1}(t) \) for \( 0 < t \leq b_1 \) is equal to \( g_{i+1}(0) + w_{i+1}(0)t \). Furthermore, let job \( j \) be the job which becomes late when \( t = b_1 \). Then function \( g_{i+1}(t) \) for \( b_1 < t \leq b_2 \) is equal to \( g_{i+1}(b_1) + w_{i+1}(b_1)(t - b_1) \). Noting that \( w_{i+1}(b_1) = w_{i+1}(0) + w_j \), we conclude that given \( g_{i+1}(b_{k-1}) \) we can calculate \( g_{i+1}(t) \) for all \( b_{k-1} < t \leq b_k \) in constant time.

**Theorem 2** The time to compute \( c(\sigma_{ij}) \) for all \( 1 \leq i < j \leq n \) is \( O(n^2) \).

**Proof.** For fixed \( i \), we can sort \( P_{i-1} + p_j \) for all \( j \) in \( O(n) \) time since the order is consistent with the order in the \( ProcessingList \). Then for all \( j \), we can determine which interval \([b_k, b_{k+1}]\) the value \( P_{i-1} + p_j \) falls in and hence evaluate \( g_{i+1}(P_{i-1} + p_j) \) with a total time of \( O(n) \).

Using the same methodology \( g_j(P_j-p_i) \) can be calculated in \( O(n) \) time for each fixed value of \( j \) and for all \( i < j \). Hence \( g_{i+1}(P_{i-1} + p_j) \) and \( g_j(P_j-p_i) \) are calculated in \( O(n^2) \) time for all \( i \) and \( j \), establishing the theorem. \( \blacksquare \)

### 2.3 The Twist Neighborhood

In this subsection, we consider the neighborhood consisting of taking a subset of the jobs \( \{i, i+1, \ldots, j-1, j\} \) and processing them in reverse order. Let \( \tau_{ij} \) be the schedule obtained from \( \pi^* \) by reversing the order of jobs \( i \) to \( j \) for \( 1 \leq i < j \leq n \). That is

\[
\tau_{ij}(k) = \begin{cases} 
  k & \text{if } k < i \text{ or } k > j \\
  j - k + i & \text{if } i \leq k \leq j.
\end{cases}
\]

We next show how to search the twist neighborhood in \( O(n^2) \) time.

The preprocessing steps required for Theorem 3 follows along similar lines to those of Section 2.2. We first sort jobs of \( \tau_{1n} \) in order of non-increasing lateness and store them in the ordered list \( RevLateList \). Let \( ProcessingList \) and \( W(i) \) be defined as before.
Furthermore, let \( h_k(t) \) be the weighted sum of tardiness if we schedule only jobs \( k, k - 1, \ldots, 1 \) and in that order, and if job \( k \) starts at time \( t \). Thus

\[
h_k(t) = \sum_{i=1}^{k} w_i (P_k - P_{i-1} + t - d_i)^+.
\]

We note that

\[
c(\tau_{ij}) - c(\pi^*) = [h_j(P_{i-1}) - h_{i-1}(P_j)] - [W(j) - W(i - 1)]. \tag{4}
\]

The first term of the right hand side of (4) is the weighted tardiness of the jobs \( i, \ldots, j \) in schedule \( \tau_{ij} \) and the last term is the weighted tardiness of the jobs \( i, \ldots, j \) in the original schedule \( \pi^* \). The weighted tardiness due to the rest of the jobs stays the same between these two schedules.

Hence to compute \( c(\tau_{ij}) \) for all \( i \) and \( j \) in \( O(n^2) \) time, it suffices to compute \( h_j(P_{i-1}) \) and \( h_{i-1}(P_j) \) for all \( i \) and \( j \) in \( O(n^2) \) steps.

We next show how to compute \( h_j(P_{i-1}) \) for a given \( j \) and for all \( i < j \) in \( O(n) \) steps. The procedure follows along very similar lines to computing the values for \( g_{i+1}(P_{i-1} + p_j) \).

We note that \( h_j(t) \) is a piecewise linear convex function of \( t \). Furthermore, its slope changes at breakpoints corresponding to the values of \( t \) for which \( d_k = P_j - P_{k-1} + t \) for some \( k \in \{1, \ldots, j\} \). The order of the breakpoints agrees with the order of the jobs on \( \text{RevLateList} \), and so one can determine the breakpoints and the linear functions on the intervals between these breakpoints, in order, in \( O(n) \) time.

For fixed \( j \), the values \( P_{i-1} \) are already in sorted order. Then for all \( i \), we can evaluate \( h_j(P_{i-1}) \) with a total time of \( O(n) \).

Furthermore, using exactly the same methodology \( h_{i-1}(P_j) \) for a given \( i \) can also be calculated in \( O(n) \) time for all \( j > i \). Hence \( h_j(P_{i-1}) \) and \( h_{i-1}(P_j) \) are calculated in \( O(n^2) \) time for all \( i \) and \( j \). We have thus established the following theorem.

**Theorem 3** The time to compute \( c(\tau_{ij}) \) for all \( 1 \leq i < j \leq n \) is \( O(n^2) \).

### 3 Extensions

In this section, we briefly show how the results of Section 2 can be used to obtain efficient algorithms for searching neighborhoods that are extensions of the insertion, swap, and twist neighborhoods. We first discuss the ‘dynasearch swap’ neighborhood which is exponentially sized but is searched in polynomial time. Similar ideas apply in the same manner to dynasearch insertion and dynasearch twists neighborhoods. Next, we establish the complexity of searching the neighborhood that consists of all schedules that are obtained by applying a set of independent insertion, swap, and twist moves to the current schedule. Finally, we demonstrate the impact of the above results on the search time of a restricted version of the swap neighborhood.
3.1 Dynasearch Swap Neighborhood

Two swaps applied to \( \pi^* \) creating schedules \( \sigma_{ij} \) and \( \sigma_{kl} \) are said to be independent if either \( j < k \) or \( l < i \). The dynasearch swap neighborhood of \( \pi^* \) consists of all schedules that are obtained by applying a set of independent swaps simultaneously. Furthermore, the total change on the weighted tardiness of \( \pi^* \) caused by a set of independent swaps is equal to the sum of the changes caused by the individual swaps in this set.

Congram et al. [1] and Ergun [2] show that the dynasearch swap neighborhood has exponential size. However searching this large scale neighborhood implicitly via dynamic programming ([1]) or network flows ([2]) techniques has the same complexity as searching the swap neighborhood. This result follows from the fact that once the cost change due to all possible swaps are evaluated, then searching the dynasearch swap neighborhood can be accomplished in \( O(n^2) \) time.

**Corollary 1** The dynasearch swap neighborhood can be searched in \( O(n^2) \) time for the \( 1||\sum w_j T_j \) problem.

**Proof.** There exist algorithms which search the dynasearch swap neighborhood in \( O(n^2) \) time given \( c(\sigma_{ij}) \) for all \( i \) and \( j \) ([1] and [2]). Hence the result follows from Theorem 2.

**Corollary 2** Given the schedule \( \pi^* \), the neighborhood that consists of all schedules that are obtained by applying a set of independent insertions, swaps, and twists to \( \pi^* \) can be searched in \( O(n^2) \) time.

**Proof.** Let \( \eta_{ij} = \min\{v_{ij}, \sigma_{ij}, \tau_{ij}\} \). By Theorems 1, 2 and 3, one can calculate \( \eta_{ij} \) for all \( i \) and \( j \) in \( O(n^2) \) time. Once the \( \eta_{ij} \) values are calculated, the best neighbor in this combined neighborhood can be found by one of the algorithms in [1] or [2].

3.2 Restricted Swap Neighborhood

Consider the neighborhood that is obtained by only allowing swaps of jobs \( i \) and \( j \) when they are at most \( K \) spaces apart. That is for \( \pi^* \) we allow swapping jobs \( i \) and \( j \) if \( j - i < K \).

**Theorem 4** Searching the neighborhood consisting of swaps limited to cases where the jobs being swapped are at most \( K \) spaces apart takes \( O(nK) \) time.

**Proof.** The preprocessing step is similar to the preprocessing step for searching over all swaps except that we form \( 2(n - K + 1) \) different ordered lists of jobs. For each \( j = 1 \) to \( n - K \), \( \text{LateList}(j) \) consists of jobs \( j, j+1, \ldots, j+K \) in non-increasing order of lateness, and \( \text{ProcessingList}(j) \) consists of jobs \( j, j+1, \ldots, j+K \) in non-decreasing order of processing time. The time to compute \( \text{LateList}(1) \) and \( \text{ProcessingList}(1) \) is \( O(K \log K) \). Furthermore, given \( \text{LateList}(j) \) and \( \text{ProcessingList}(j) \) it only takes \( O(K) \) time to compute \( \text{LateList}(j+1) \) and \( \text{ProcessingList}(j+1) \) since these lists contain the same jobs except for job \( j \) replaced by job \( j+K+1 \). One can maintain these \( O(n-K+1) \) lists implicitly or explicitly. In each case, the total time required for the preprocessing step is \( O(nK) \).
One can calculate $c(\sigma_{ij})$ for all $i$ and $j$ such that $j - i < K$ as in Section 2.2. The total running time is $O(K)$ to compute each piecewise linear function of $K$ pieces. Hence the neighborhood is searched in $O(nK)$ time plus the preprocessing time. ■

As a further speed up, one can take advantage of the fact that the sorted lists are very similar from one iteration to the next. Hence the preprocessing step takes $O(n \log n)$ time in the first iteration, and the lists can be updated in $O(n)$ time in the subsequent iterations.

Finally, we point out that all the results for the problem $1||\sum w_j T_j$ presented in this paper can be directly extended to single machine total tardiness problems with deadlines.

Acknowledgment

We thank Dan Stratila for comments leading to an improvement in the exposition. We thank an anonymous referee for pointing out the extension to the problems with deadlines and other comments which improved Theorem 4.

References


