# A fast parallel dynasearch algorithm for some scheduling problems

Wojciech Bożejko
Wrocław University of Technology
Institute of Computer Engineering, Control and Robotics
Janiszewskiego 11-17, 50-372 Wrocław, Poland
wojciech.bozejko@pwr.wroc.pl

Mieczysław Wodecki
University of Wrocław
Institute of Computer Science
Przesmyckiego 20, 51-151 Wrocław, Poland
mwd@ii.uni.wroc.pl

## Abstract

*In this paper we propose a parallel dynasearch algorithm to solve the single machine total weighted tardiness problem. Nowadays dynasearch method is the most powerful heuristic method for the single machine total weighted scheduling problem. The neighborhood is generated by executing series of swap moves. Using dynamic programming such a neighborhood, which has exponential size, is explored in polynomial time. Here we propose parallel dynasearch algorithm for the parallel mainframe computers which significantly improves results of the sequential dynasearch algorithm.*

## 1 Introduction

Finding good solutions of NP-hard combinatorial problems is generally difficult. For a large size of problem, exact algorithms may not be useful since they are computationally too expensive. Approximate algorithms, which not guarantee to find an optimal solution, are a good alternative. Popular are metaheuristics based on descent search method: simulated annealing, taboo search and its multi-start approaches.

A well-known method of improving the solution quality of descent search algorithms is adopt a multi-start approach in which several independent runs are performed, and the best of the resulting solutions is selected. However, more effective approach is to allow runs by generating the new starting solution from one of the previous local optima by a perturbation method. Such an approach is known as iterated descent search and is widely-recognized method of obtaining high solution quality at relatively low computational cost.

In traditional algorithms the neighborhood is generated by single transformations (moves). Insert type move generates the neighborhood of $n(n-1)$ elements and the swap type moves of $n(n-1)/2$ elements, where $n$ is the number of elements. In paper by Congram et al. [3] neighborhood has exponential number of elements. Each element is generated by executing series of independent moves. Applying dynasearch method it is possible to revise this neighborhood (calculate the minimal element) in polynomial time. The methods of dynamical programming require, however, a lot of time and memory, so it can be applied only in limited range of large problems.

In this paper we present parallel dynasearch algorithm in which the fundamental element is parallel method of generating and revising the neighborhood. We proved a certain properties which indicated that such a method is cost-optimal with efficiency $O(1)$. Such an algorithm we applied to solve the single machine total weighted tardiness problem. The problem is (unary) NP-hard. We use the same test instances as [3] drawn from the OR-library [2]. Obtained solutions we compare with the best known published in papers Congram et al. [3] and Grosso et al. [8].

The remainder of the paper is organized as follows. In Section 2 we give some notations and literature of the classic one machine total weighted tardiness problem. Survey of dynasearch neighborhoods which are searchable in polynomial time for some other NP-hard scheduling problems is given in the Section 3 and 3.1. Section 4 presents the method of cost-optimal parallelization of the dynasearch method. Computational experiments are given in Section 5.



Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)
0-7695-2554-7/06 $20.00 © 2006 IEEE

## 2 Single machine total weighted tardiness problem

In the single machine total weighted tardiness problem (*TWTP*), a *set of jobs* $\mathcal{N} = \{1, 2, ..., n\}$ has to be processed without interruption on a single machine that can handle only one job at a time. All jobs become available for processing at time zero. Each job $i \in \mathcal{N}$ has an integer *processing time* $p_i$, a *due date* $d_i$, and a *positive weight* $w_i$. For a given sequence of the jobs (earliest) completion time $C_i$, the *tardiness* $T_i$= $\max\{0, C_i - d_i\}$ and the cost $f_i(C_i) = w_i \cdot T_i$ of job $i \in \mathcal{N}$ can be computed. The goal is to find a job sequence with minimize the sum of the costs given by $\sum_{i=1}^{n} f_i(C_i) = \sum_{i=1}^{n} w_i \cdot T_i$.

Each schedule of jobs can be represent by a permutation $\pi = (\pi(1), \pi(2), \ ... \ , \pi(n))$. The total cost is

$$F(\pi) = \sum_{i=1}^{n} f_{\pi(i)}(C_{\pi(i)}),$$

where completion time of job $\pi(i)$, $C_{\pi(i)}) = \sum_{j=1}^{i} p_{\pi(i)}$.

Graham et al. [9] have proposed an efficient notation where each problem is described by a record composed of three fields $\alpha|\beta|\gamma$. The first field $\alpha$ identifies the number of machines. The second field $\beta$ describes a list of special characteristics of the jobs or the environment, and last field $\gamma$ identifies the cost function.

The single machine total weighted tardiness problem is represented in the literature by $1||\sum_i w_i T_i$ and it belongs to a strong NP-hard class (Lawler [10], Lenstra et al. [11]). A large number of studies has been devoted to the problem. Emmons [6] proposes several dominance rules that restrict the search process for an optimal solution. These rules are used in many algorithms. Enumerative algorithms that use dynamic programming and branch and bound approaches to the problem are described by Fischer [7], Potts and Van Wassenhove [13]. These and other algorithms are discussed and tested in a review paper by Abdul-Razaq et al. [1]. Algorithms consitute a significant improvement to the exhaustive search, but they remain laborious and are applicable only to relatively small problems (with the number of jobs not exceeding 50). The enumerative algorithms require considerable computer resources both in terms of the computation times and the core storage. Therefore, many algorithms have been proposed to find near optimal schedules in a reasonable time. These algorithms can be broadly classified into construction and improvement methods.

The construction methods use dispatching rules to come up with a solution by fixing a job in a position at each step. Several constructive heuristics are described by Fischer [7] and in a review paper by Potts and Van Wassenhove [12]. They are very fast, but their quality is not good.

The improvement methods start from an initial solution and repeatedly try to improve the current solution by local changes. The interchanges are continued until a solution that cannot be improved is obtained. Such a solution is a local minimum. To increase the performance of local search algorithms, there are used metaheuristics like Tabu Search (Crauwels et al. [4]), Simulated Annealing (Potts and Van Wassenhove [12]), Genetic Algorithms (Crauwels et al. [4]), Ant Colony Optimization (Den Basten et al. [5]). A very effective local-search method was proposed by Congram et al. [3] and next improved by Grosso et al. [8]. The key aspect of the method is its ability to explore an exponential-size neighborhood in polynomial time, using a dynamic programming technique.

## 3 The dynasearch neighborhood

Let $(a, b)$ be a pair of position in a permutation $\pi = (\pi(1), \pi(2), \ ... \ , \pi(n))$, $1 \leq a, b \leq n$. I the *swap move* jobs $\pi(a)$ and $\pi(b)$ are swapped in some positions $a$ and $b$ in $\pi$. Now, the move $v = (a, b)$ generates permutation $\pi^v$ from $\pi$ as follows:

$$\pi^v = (\pi(1), \ ... \ , \pi(a-1), \pi(b), \pi(a+1), \ ...$$

$$..., \ \pi(b-1), \pi(a), \pi(b+1), \ ... \ , \ \pi(n)).$$

The *swap neighborhood* of $\pi$ consists of permutations $\pi^v$ generated by swap moves $v = (a, b)$, $1 \leq a, b \leq n$ . The neighborhood has size $n(n-1)/2$. For further considerations, we refer to the notions and notations taken from the papers Congram et al. [3].
The two swap moves $v = (i, j)$ and $u = (k, l)$, are said to be *independent* if

$$\max\{i, j\} < \min\{l, \ k\} \text{ or } \min\{i, j\} > \max\{l, \ k\}.$$

The *dynasearch swap neighborhood* consists of all permutation that can be obtained from $\pi$ by a series of pairwise independent swap moves. The size of the neighborhood is $2^{n-1} - 1$.

We define a partial sequence to be in state $(j, \pi)$, for $j = 1, 2, \ ... \ , n$, if it can be obtained from the partial sequence $\pi(1), \pi(2), \ ... \ pi(j)$ by applying a series of independent swops. Let $\pi_j$ be a partial sequence with minimum total weighted tardiness for jobs $\pi(1), \pi(2), ..., \pi(j)$ among partial sequences in state $(j, \pi)$. Further, let $F(\pi_j)$ be the total weighted tardiness for jobs $\pi(1), \pi(2), ..., \pi(j)$ in $\pi_j$ i.e.

$$F(\pi_j) = min\{F(\beta) : \quad \beta \in (j, \pi\}.$$

This partial sequence must be obtained from a partial sequence $\pi_i$ that has minimum objective value from all partial sequences in first previous state $(i, \pi)$, where $0 \leq i < j$, by appending job $\pi(j)$ if $i = j - 1$, or by first appending



Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)
0-7695-2554-7/06 $20.00 © 2006 IEEE

job $\pi(j)$ and then interchanging jobs $\pi(i+1)$ and $\pi(j)$ if $0 \leq i < j-1$. These two possibilities are considered in detail bellow.

**A.** $i = j-1$. In this case, job $\pi(j)$ is not involved in any $\pi(j)$ swap, and $\pi(j)$ is simply append to a partial sequence $\pi_{j-1}$; hence $\pi_j = (\pi_{j-1}, \pi_j)$. Accordingly,

$$F(\pi_j) = F(\pi_{j-1}) + w_{\pi(j)}(C_{\pi(j)} - d_{\pi(j)})^+,$$

where, for any real $x$, $(x)^+ = \max\{0,\ x\}$.

**B.** $0 \leq i < j-1$. Here, jobs $\pi(j)$ and $\pi(i+1)$ are swapped, so that $\pi_j$ can be written as $\pi_j = (\pi_i, \pi(j), \pi(j+2), .., \pi(j-1), \pi(i+1))$, and the total weighted tardiness $F(\pi_j)$ is readily computed as

$$F(\pi_j) = F(\pi_i) + w_{\pi(j)}(C_{\pi(i)} + p_{\pi(j)} - d_{\pi(j)})^+ + \\ + \sum_{k=i+2}^{j-1} w_{\pi(k)}(C_{\pi(k)} + p_{\pi(j)} - p_{\pi(i+1)} - d_{\pi(j)})^+ + \\ w_{\pi(i+1)}(C_{\pi(j)} - d_{\pi(i+1)})^+.$$

These values can be determined recursively using dynamic programming.

For any $j = 2,\ 3\ ,\ \ldots\ ,\ n$

$$F(\pi_j) = \min \begin{cases} F(\pi_{j-1}) + w_{\pi(j)}(C_{\pi(j)} - d_{\pi(j)})^+, \\ \min\limits_{0 \leq i \leq j-2} \{F(\pi_i) + w_{\pi(j)}(C_{\pi(i)} + p_{\pi(j)} - \\ -d_{\pi(j)})^+ + \sum\limits_{k=i+2}^{j-1} w_{\pi(k)}(P_{\pi(k)} + p_{\pi(j)} - \\ -p_{\pi(i+1)} - d_{\pi(k)})^+ + \\ + w_{\pi(i+1)}(C_{\pi(j)} - d_{\pi(i+1)})^+\}. \end{cases} \quad (1)$$

where, the initialization is $F(\pi_0) = 0$ and $F(\pi_1) = w_{\pi(1)}(p_{\pi(1)} - d_{\pi(1)})^+$.

The optimal solution is $F(\pi_n)$ and this dynamic programming algorithm runs in $T_{seq} = O(n^3)$ time.

## 3.1 Dynasearch method for some one machine scheduling problems

In this section we present application of the dynasearch method for some NP-hard single machine scheduling problems.

### 3.1.1 Problem $1|r_i|\sum w_i T_i$

In this problem a starting time $C_i$ of a job $i \in \mathcal{N}$ is not less that release date $r_i$. For any permutation (sequence) of jobs $\pi = (\pi(1), \pi(2),\ \ldots\ , \pi(n))$, a time of finishing of the job $\pi(i) \in \mathcal{N}$

$$C_{\pi(i))} = C_{\pi(i-1)} + (r_{\pi(i)} - C_{\pi(k-1)})^+ + p_{\pi(i)},$$

where $C_\pi(0) = 0$.
Recurrence formula (1) takes form of:

$$F(\pi_j) = \min \begin{cases} F(\pi_{j-1}) + w_{\pi(j)}(C_{\pi(j)} - d_{\pi(j)})^+, \\ \min\limits_{1 \leq i \leq j-1} \{G(\pi_i)\}. \end{cases} \quad (2)$$

where, the initialization is $F(\pi_0) = 0$ and $F(\pi_1) = w_{\pi(1)}(C_{\pi(1)} - d_{\pi(1)})^+$.
The value of the $G(\pi_i)$ function can be computed from the following procedure:

```
G(j) {
    ret = F(j-1);
    tmp = max{P(j-1), r_i};
    ret+= max{(tmp + p_i - d_i),0}*w_i;
    tmp+=p_i;
    for(v = j+1; v <= i-1; v++)
    {
        tmp=max{tmp,r_v};
        ret+=max{(tmp + p_v - d_v),0}*w_v;
        tmp+=p_v;
    }
    tmp=max{tmp, r_j};
    ret+=max{(tmp + p_j - d_j), 0}*w_j;
    return ret;
}
```

Computational complexity of this procedure is $O(j^2)$. Therefore determining the value of $F(\pi_n)$ from the formula (2) has complexity $O(n^3)$.

### 3.1.2 Problem $1||\sum w_i U_i$

For a given schedule $\pi$ we can compute for job $\pi(i)$ its completion time $C_{\pi(i)}$ and the unit penalty $U_{\pi(i)}$. $U_{\pi(i)} = 1$ if $C_{\pi(i)} > d_{\pi(i)}$ and $U_{\pi(i)} = 0$ otherwise. In another form of notation $U_{\pi(i)} = sgn((C_{\pi(i)} - d_{\pi(i)})^+)$.

Therefore $G(i) = F(i-1) + sgn((P(i) - d_i)^+) \cdot w_i$. For $j < i$ we can compute similarly as in original *dynasearch* method:

$$F(\pi_j) = \min \begin{cases} F(\pi_{j-1}) + sgn((C_{\pi(j)} - d_{\pi(j)})^+) w_{\pi(j)}, \\ \min\limits_{0 \leq i \leq j-2} \{F(\pi_{j-1}) + sgn((C_{\pi(j-1)} + \\ +p_{\pi(i)} - d_{\pi(i)})^+) w_{\pi(i)} + sgn((C_{\pi(i)} - \\ -d_{\pi(j)})^+) w_{\pi(j)} + \sum\limits_{k=i+2}^{j-1} sgn((C_{\pi(k)} - \\ -p_{\pi(j)} + p_{\pi(i)} - d_{\pi(k)})^+) w_{\pi(k)}. \end{cases}$$

Computational complexity of search over all the neighborhood is $O(n^3)$. From (2) and (3) it is easy to formulate recurrence relationship for the $1|r_i|\sum w_i U_i$ problem.



Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)
0-7695-2554-7/06 $20.00 © 2006 IEEE

#### 3.1.3 Problem $1||\sum(w_iE_i + u_iT_i)$

In this problem by $e_i$ and $d_i$ we mean an adequately demanded earliest and latest moment of the finishing processing of a job $i \in \mathcal{N}$,. If a scheduling of jobs is established and $C_i$ is the moment of finishing of a job $i$, then we call $E_i = \max\{0, e_i - C_i\}$ an *earliness* and $T_i = \max\{0, C_i - d_i\}$ a *tardiness*. The expression $u_iE_i + w_iT$ is a *cost* of execution a job, where $u_i$ and $w_i$ ($i \in \mathcal{N}$) are nonnegative coefficients of a goal function. The problem consists in minimizing a sum of costs of jobs, that is the function $\sum_{i=1}^{n}(u_iE_i + w_iT_i)$.

To obtain right recurrence relationship it is necessary to exchange the cost of job $w_iT_i$ for $u_iE_i+w_iT_i$ in the formula (1).

#### 3.1.4 Problem $1||\sum w_iC_i$

For the problem of minimize costs of finishing of jobs the recurrence formula (1) takes the form of:

$$F(\pi_j) = \min \begin{cases} F(\pi_{j-1}) + w_{\pi(j)}C_{\pi(j)}, \\ \min\limits_{0\le i\le j-2}\{F(\pi_i) + w_{\pi(j)}(C_{\pi(i)}+ \\ +p_{\pi(j)}) + \sum\limits_{k=i+2}^{j-1} w_{\pi(k)}(C_{\pi(k)}+ \\ +p_{\pi(j)} - p_{\pi(i+1)} + w_{\pi(i+1)}C_{\pi(j)}\}. \end{cases}$$

The algorithm of determining optimal value of $F(\pi_n)$ has computational complexity $O(n^3)$.

## 4 Parallel dynasearch method

**Fact 1** *Sequence of prefix sums $(y_1, y_2, ..., y_n)$ of input sequence $(x_1, x_2, ..., x_n)$ such, that*

$$y_k = y_{k-1} + x_k = x_1 + x_2 + ... + x_k \text{ for } k = 2, 3, ..., n$$

*where $y_1 = x_1$ can be calculated in time $O(\log n)$ on EREW PRAM machine with $O(n/\log n)$ processors.*

From this fact we have immediately, that the sum of $n$ values can be calculated in time $O(\log n)$ on $O(n/\log n)$ – processors EREW PRAM machine.

**Fact 2** *Minimal and maximal value of input sequence $(x_1, x_2, ..., x_n)$ can be determined in time $O(\log n)$ on EREW PRAM machine with $O(n/\log n)$ processors.*

If we have not such a big number of processors, we can use such a fact to keep the same cost:

**Fact 3** *If algorithm $A$ works on $p$ – processors PRAM in time $t$, so for every $p' < p$ exists an algorithm $A'$ for the same problem, which works on $p'$ – processors PRAM in time $O(pt/p')$.*

Now we can formulate a theorem of dynasearch parallelization.

**Theorem 1** *The best element of the dynasearch neighborhood can be determined in time $O(n\log^2 n)$ on the PRAM machine with $O(\frac{n^2}{\log^2 n})$ processors.*

**Proof.** Let us notice, that all times of job's finishing $C_j$, $j$ = 1,2,...,$n$ can be computed as prefix sums (see Fact 1) in time $O(\log n)$ using $O(n/\log n)$ processors. Next, to compute the value of $F(\pi_j)$ in the equation (1) it is necessary to determine the sum of at the most $n$ values of $\sum\limits_{k=i+2}^{j-1} w_{\pi(k)}(P_{\pi(k)} + p_{\pi(j)} - p_{\pi(i+1)} - d_{\pi(k)})^+$ a next the minimal element (at most) $n$ values $\min\limits_{0\le i\le j-2}\{F(\pi_i) + w_{\pi(j)}(C_{\pi(i)}+p_{\pi(j)}-d_{\pi(j)})^+ + \sum\limits_{k=i+2}^{j-1} w_{\pi(k)}(P_{\pi(k)}+p_{\pi(j)} - p_{\pi(i+1)} - d_{\pi(k)})^+ + w_{\pi(i+1)}(C_{\pi(j)} - d_{\pi(i+1)})^+\}$, with counted sums inside. Operations of addition, by its small and - first of all - constant and independent from the $n$ number, can be executed in the constant time $O(1)$, so we can pass it over in reasoning. To determine mentioned minimal element we need PRAM machine with $O(\frac{n}{\log n})$ processors (see Fact 2), from which every one have to execute computations of (mentioned on the beginning) sums also with $O(\frac{n}{\log n})$ processors, so in total we need $O(\frac{n}{\log n}) \cdot O(\frac{n}{\log n}) = O(\frac{n^2}{\log^2 n})$ processors of the PRAM oraz machine and time $T_{par} = O(\log n) \cdot O(\log n) = O(\log^2 n)$.

On the end, it is necessary to compute the minimum of two counted values of (1), what we can do in the constant time $O(1)$ on one processor. Therefore to determine all values $F(\pi_j)$, $j = 2, 3, \ldots, n$ we need time $T_{par} = n \cdot O(\log^2 n) = O(n\log^2 n)$ and $p = O(\frac{n^2}{\log^2 n})$ processors.

The conclusion of this theorem is following: such a method has a speedup

$$S = \frac{T_{seq}}{T_{par}} = O(\frac{n^3}{n\log^2 n}) = O(\frac{n^2}{\log^2 n})$$

in the order of the number of used processors, so the algorithm connected with this theorem is cost-optimal with the efficiency $O(1)$. Obtained speedup has asymptotically maximal possible value.



Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)
0-7695-2554-7/06 $20.00 © 2006 IEEE

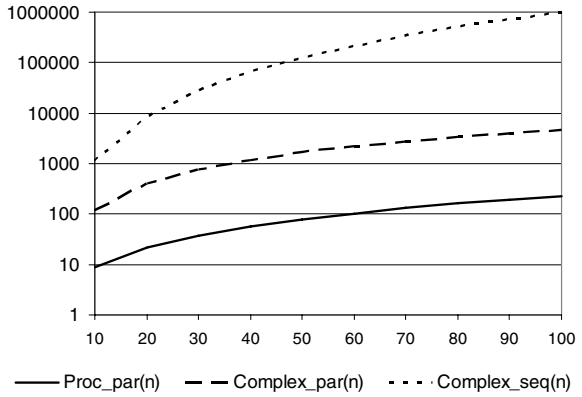


**Figure 1. Comparison (in logarithmic scale) of speed of complexity functions increasing.**

Parallelization gives us two possible benefits: shorten of computations time or examine more solutions in the same time, because Fact 3 allows search dynasearch neighborhood i.e. with 4 processor PRAM machine in time $O(n^3/4)$, so 4 times faster than sequential algorithm does. It is also possible to use all $O(\frac{n^2}{\log^2 n})$ processors, keeping cost optimality and obtaining maximal speedup of computations process. Mentioned number of processors is not potentially too big and can be met in real parallel systems. Comparison of speed of functions increasing (complexity of sequential and parallel method, number of processors) is presented on Figure 1, where $\mathrm{Complex\_seq}(n) = n^3$, $\mathrm{Complex\_par}(n) = \frac{n}{\log^2 n}$, $\mathrm{Proc\_par}(n) = \frac{n^2}{\log^2 n}$.

## 5 Computational results

The algorithm was implemented in Ada95 language and run on 4 – processors Sun Enterprise 4 x 400 MHz under Solaris 7 operating system. Tasks of Ada95 language were executed in parallel as system threads. An implementation of the parallel dynasearch algorithm was tested on problems with $n$=40, 50 and 100 jobs of benchmark instances taken from the OR-library [2]. The benchmark set contains 125 instances for each size of $n$ value. There are obtained results compared with the best known results from literature in Table 1 and Table 2. For comparison, also the results of the constructive algorithm META (composition of SWPT, EDD, AU and COVERT algorithms) from [12] are presented. The chosen measure was relative distance (in percent) to the best known solution's goal function. Number of optimal obtained values (NO) is also presented.

As we can see in Table 1 the average distance to reference solutions for sequential and parallel dynasearch algorithms was at the level of $0.297\%$ for 1-processor implementation and at the level of $0.153\%$ for 4-processors implementation. Number of optima found was in average 79 for the parallel dynasearch implementation and was $18\%$ greater than NO of the sequential algorithm. Times of executing were comparable.

| $n$ | 1 processor | | |
|---|---|---|---|
| | %diff | NO | time (sec.) |
| 40 | 0.282 | 92 | 0.112 |
| 50 | 0.256 | 73 | 0.256 |
| 100 | 0.354 | 36 | 0.632 |
| **Average** | **0.297** | **67** | **0.333** |

**Table 1. Sequential dynasearch algorithm.**

| $n$ | 4 processors | | | META |
|---|---|---|---|---|
| | %diff | NO | time (sec.) | %diff |
| 40 | 0.043 | 110 | 0.116 | 15.919 |
| 50 | 0.124 | 85 | 0.260 | 14.692 |
| 100 | 0.291 | 42 | 0.640 | 16.640 |
| **Average** | **0.153** | **79** | **0.339** | **15.750** |

**Table 2. Parallel dynasearch algorithm and META.**

## 6 Conclusions

Iterated dynasearch algorithm can be easily parallelized by concurrent run of search threads, connected by common list of the best known local optima. Additionally, method connected with Theorem 1 gives effective method of cost-optimal parallelization. In sucha a way it is possible to take advantage of full computational power of parallel mainframe computers to find good solutions of hard scheduling problems.

## References

[1] T.S. Abdul-Razaq, C.N. Potts, L.N. Van Wassenhove, A survey of algorithms for the single machine total weighted tardiness scheduling problem, Discrete Applied Mathematics, 26, 1990, 235-253.

[2] Beasley J.E., OR-Library: distributing test problems by electronic mail, Journal of the Operational Research Society 41, 1990, 1069-1072.

[3] Congram K.R., C.N. Potas, S. Van de Velde, An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, INFORMS Journal on Computing, 14, 2002, 52-67.



Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)
0-7695-2554-7/06 $20.00 © 2006 IEEE

[4] Crauwels H.A.J., Potts C.N., Van Wassenhove L.N., Local Search Heuristics for the Single machine Total Weighted Tardiness Scheduling Problem, INFORMS Journal on Computing, Vol. 10, No. 3, 1998, 341-350.

[5] Den Basten, M., T. Stützle, M. Dorigo, Design of Iterated Local Search Algoritms An Example Application to the Single Machine Total Weighted Tardiness Problem, J.W.Boers et al. (eds.) Evo Worskshop 2001, LNCS 2037, 2001, 441-451.

[6] H. Emmons, One machine sequencing to Minimize Certain Functions of Job Tardiness, Operations Research, 17, 1969, 701-715.

[7] M.L. Fisher, A Dual Algorithm for the One Machine Scheduling Problem, Mathematical Programming, 11, 1976, 229-252.

[8] Grosso A., F. Della Croce, R. Tadei, An enhanced dynasearch neighborhood for the single-machine total weighter tardiness scheduling problem, Operations Research Letters, 32, 2004, 68-72.

[9] Graham M.R., E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, Optimization an approximation in detrministic sequencing and scheduling: a survey, Annals of Discrete Mathematics, 3, 1979, 287-326.

[10] Lawler, E.L., A pseudopolynomial Algorithm for Sequencing Jobs to Minimize Total Tardiness, Annals of Discrete Mathematics, 1, 1977, 331-342.

[11] Lenstra, J.K., Rinnoy Kan, A.G.H., and P.Brucker, Complexity of Machine Scheduling Problems, Annals of Discrete Mathematics, 1, 1977, 343-362.

[12] Potts C.N., Van Wassenhowe L.N., Single Machine Tardiness Sequencing Heuristics, IIE Transactions, 23, 1991, 346-354.

[13] C.N. Potts, C.N. Van Wassenhove, A Branch and Bound Algorithm for the Total Weighted Tardiness Problem, Operations Research, 33, 1985, 177-181.



Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)
0-7695-2554-7/06 $20.00 © 2006 IEEE