Available online at www.sciencedirect.com
SCIENCE DIRECT®
Operations Research Letters 32 (2004) 68–72
Operations Research Letters
www.elsevier.com/locate/dsw

An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem

A. Grosso[a], F. Della Croce[b,*], R. Tadei[b]

[a] Dipartimento di Informatica, Università di Torino Corso Svizzera 185, 10149 Torino, Italy
[b] Dipartimento di Automatica e Informatica, Politecnico di Torino Corso Duca degli Abruzzi 24, 10129 Torino, Italy

Received 3 September 2002; accepted 16 April 2003

Abstract

Based on the work by Congram, Potts and Van de Velde, we develop for the single-machine total weighted tardiness scheduling problem an enhanced dynasearch neighborhood obtained by the generalized pairwise interchange (GPI) operators. Despite of the wider neighborhood considered, a fast search procedure using also elimination criteria is developed. The computational results significantly improve over those of Congram, Potts and Van de Velde.

Keywords: Total weighted tardiness; Dynasearch; Generalized pairwise interchanges

1. Introduction

In the single-machine total weighted tardiness problem denoted $1||\sum w_iT_i$ a set of jobs $N=\{1,2,\ldots,n\}$ is given, where each job i has integer processing time p_i, due date d_i and weight w_i. The problem calls for finding a sequence S and related completion times C_i minimizing $T(S)=\sum_{i=1}^{n} w_iT_i$, with $T_i=(C_i-d_i)^+=\max(C_i-d_i,0)$.

The $1||\sum w_iT_i$ problem is NP-hard in the strong sense [3] and the existing exact algorithms can solve to optimality instances with moderate size only. On the other hand, the problem has been successfully tackled by local search heuristic algorithms where the neighborhood of a solution is usually computed by applying swap (a.k.a. pairwise interchange) operators to the working sequence.

Dynasearch is a neighborhood search technique introduced in [4] whose main feature is the ability of searching exponential size neighborhoods in polynomial time by exploiting problem structure. To the authors' knowledge, the so-called *iterated* dynasearch algorithm for the $1||\sum w_iT_i$ problem proposed in [1] has given the best-known results in terms of solution quality and computation times.

Generalized pairwise interchanges (GPI) extend the common swap neighborhood by allowing also job insertion moves, and have been shown to provide an effective neighborhood for several scheduling problems [2].

In this paper, we develop a GPI-based dynasearch neighborhood which widens the swap-based one of [1] (where such extension was mentioned but not

* Corresponding author. Tel.: +39-011-564-7059; fax: +39-011-564-7099.
E-mail addresses: grosso@di.unito.it (A. Grosso), federico.dellacroce@polito.it (F. Della Croce), roberto.tadei@polito.it (R. Tadei).

0167-6377/03/$ - see front matter
doi:10.1016/S0167-6377(03)00064-6

investigated). We utilize elimination rules to efficiently search this wider neighborhood showing that a significantly better solution quality can be achieved, in about the same CPU time.

2. The GPI dynasearch neighborhood

Consider any initial sequence S. Renumber the jobs such that $S = (1, 2, \ldots, n)$. Following [2], we define four GPI operators, namely API (adjacent pairwise interchange), NAPI (nonadjacent pairwise interchange), EBSR (extraction and backward-shifted reinsertion) and EFSR (extraction and forward-shifted reinsertion) which act on the sequence as follows. Given a pair of indices $i < j$, such that $S = \sigma i \pi j \omega$:

API(i,j): $\sigma i j \omega \Rightarrow \sigma j i \omega$ (requires $\pi = \emptyset$),

NAPI(i,j): $\sigma i \pi j \omega \Rightarrow \sigma j \pi i \omega$,

EBSR(i,j): $\sigma i \pi j \omega \Rightarrow \sigma j i \pi \omega$,

EFSR(i,j): $\sigma i \pi j \omega \Rightarrow \sigma \pi j i \omega$.

API and NAPI operators are grouped together in the so-called SWAP operator applied in [1]. Two GPIs acting on position pairs $i < j$ and $k < l$ are called *independent* if $j < k$ or $l < i$.

A dynasearch procedure searches for the most profitable series of independent interchanges, thus allowing several improving steps to be performed simultaneously. Congram et al. [1] showed that for the SWAP operator the $1||\sum w_i T_i$ problem structure allows the set of possible interchange series to be searched in $O(n^3)$ time by means of a dynamic programming recursion.

We develop a GPI-based dynasearch neighborhood as follows. We start from the initial sequence S with completion times $C_j = \sum_{s=1}^{j} p_s$. At a generic stage j of the recursion, we consider the sequence $\sigma_j = ([1], \ldots, [j])$ defined as the sequence with minimum total weighted tardiness $f(j)$ among those reachable from $(1, \ldots, j)$ through a series of independent interchanges. When building the optimal sequence σ_j, one can either just append job j to σ_{j-1} or append j and execute a GPI with some position $i < j$. Taking into account these two alternatives, the dynasearch recursion can be stated, for the GPI operators $\gamma = \text{SWAP}$, EBSR, EFSR and indices $j = 0, 1, \ldots, n$, as

$$f(0) = 0, \tag{1}$$

$$f(1) = w_1(p_1 - d_1)^+, \tag{2}$$

$$f(j) = \min \left\{ f(j-1) + w_j(C_j - d_j)^+, \min_{1 \leqslant i \leqslant j-1;\gamma} \{ f(i-1) + I^{\gamma}(i,j) \} \right\}, \quad j = 2, \ldots, n, \tag{3}$$

where $I^{\gamma}(i,j)$ is the total weighted tardiness of the partial sequence $(i, \ldots, j)$ under application of the considered GPI operator γ. $I^{\gamma}(i,j)$ is set as depicted in Fig. 1. The best sequence of interchanges is found for $j = n$. If only the operator $\gamma = \text{SWAP}$ is considered, the recursion of Congram et al. [1] is obtained. Evaluating a single SWAP, EBSR or EFSR requires at most $O(n)$ time; no more than j job interchanges need to be considered at each stage, and n stages have to be solved, hence evaluating the GPI neighborhood preserves the $O(n^3)$ time bound computed for the SWAP neighborhood.

$$I^{\gamma}(i,j) = \begin{cases} w_i(C_j - d_i)^+ + w_j(C_i - p_i + p_j - d_j)^+ + \sum_{l=i+1}^{j-1} w_l(C_l - p_i + p_j - d_l)^+, & \gamma = \text{SWAP}, \\ w_j(C_i - p_i + p_j - d_j)^+ + \sum_{l=i}^{j-1} w_l(C_l + p_j - d_l)^+, & \gamma = \text{EBSR}, \\ w_i(C_j - d_i)^+ + \sum_{l=i+1}^{j} w_l(C_l - p_i - d_l)^+, & \gamma = \text{EFSR}. \end{cases}$$

Fig. 1. Values for $I^{\gamma}(i,j)$.

In order to speed up the search, the evaluation of an interchange between job i and j can be abandoned as soon as it proves to be nonoptimal. We use the following rules.

Elimination rule 1. *The* SWAP *operator between i and j does not need to be applied if at least one of the following conditions holds.*

(a) $p_i < p_j, w_i > w_j$ *and* $\max(C_i - p_i + p_j, d_j) \geqslant d_i$,
(b) $w_i \geqslant w_j$, $d_i \leqslant d_j$ *and* $d_j + p_j \geqslant C_j$,
(c) $C_j \leqslant d_j$.

Proof. The validity of the rule is based on known results about the $1|\ |\sum w_i T_i$ problem. The SWAP operator is proved to be nonoptimal if we can find a different series of independent GPIs which leads to a job sequence with lower tardiness. At stage j, consider any sequence $S = \sigma i \pi j \omega$. In [5] it is proven that: condition (a) implies $T(\sigma j \pi i \omega) \geqslant T(\sigma i \pi j \omega)$, conditions (b) and (c)—separately—imply $T(\sigma j \pi i \omega) \geqslant T(\sigma \pi i j \omega)$. Then

- Condition (a) implies that the corresponding SWAP is nonoptimal.
- Condition (b) implies that sequence $\sigma \pi i j \omega$, obtained by applying EFSR between i and $j-1$, is not worse than $\sigma j \pi i \omega$. Hence, the considered SWAP is dominated.
- Similar arguments hold for condition (c). □

Elimination rule 2. *The* EBSR *operator between i and j does not need to be applied if at least one of the following conditions holds.*

(a) $T(\sigma j i \pi \omega) < T(\sigma i j \pi \omega)$,
(b) $C_j \leqslant d_j$.

Proof. Condition (a) implies that, with $\sigma i \pi j \omega$ as initial sequence, the EBSR between i and j is dominated by the one between $i+1$ and j; the proof for (b) is trivial, since j is early. □

All the above-listed conditions can be checked in constant time. Note that only condition 1(a) can be checked in the SWAP neighborhood of [1].

By using $\gamma = \text{EBSR, EFSR}$ only, the time bound for the recursion reduces to $O(n^2)$ since the values for $I^{\gamma}(i,j)$ can then be computed in an incremental way:

$$I^{\text{EBSR}}(i-1,j) = I^{\text{EBSR}}(i,j) + \min\{(C_i - p_i + p_j - d_{i-1})^+, p_j\} - \min\{(C_i - p_i + p_j - d_j)^+, p_{i-1}\},$$

$$I^{\text{EFSR}}(i,j+1) = I^{\text{EFSR}}(i,j) + \min\{(C_j + p_{j+1} - d_i)^+, p_{j+1}\} - \min\{(C_j + p_{j+1} - d_{j+1})^+, p_i\}.$$

The basic ingredients of the recursion can be combined in various ways. For example, we can consider eliminating the SWAP operator from the neighborhood, but computational experiments showed that this would significantly reduce the quality of the final solution. Based on such experiments, we chose the following implementation: $I^{\text{EFSR}}(i,j)$ is precomputed incrementally before the recursion starts (thus allowing an initial $O(n^2)$ overhead), whilst the values of $I^{\text{SWAP}}(i,j)$ and $I^{\text{EBSR}}(i,j)$ are computed directly inside the recursion since for such operators the elimination rules are efficient, keeping the $O(n^3)$ procedure quite fast.

As a further speedup, consider the following. If an upper bound Δ on $f(j)$ is known, evaluation of a SWAP or EBSR can be abandoned as soon as it proves to lead to a partial sequence with total weighted tardiness $W \geqslant \Delta$. Congram et al. [1] use simple lower bounds on W for a priori checking of $W \geqslant \Delta$ in their SWAP neighborhood. By precomputing $I^{\text{EFSR}}(i,j)$, we obtain a better value of Δ which allows for further GPIs elimination.

3. Computational results

In order to assess the behavior of the neighborhood, we set it in the same iterated dynasearch approach developed in [1]—to which we refer for a detailed description—except for the procedure for transposing adjacent nonlate jobs during the so-called "kick" (i.e., perturbation of the local minimum by means of random SWAPs). Indeed, preliminary experiments with the GPI neighborhood gave better results without such a procedure. We use the same test instances drawn from the OR-library (see http://www.ms.ic.ac.uk/info.html) where 125

Table 1
Comparison between SWAP-based dynasearch and GPI-based dynasearch

	ND	NOPT		T_{avg}	
		SWAP	GPI	SWAP	GPI
$n = 40$	50	123.52	124.76	0.01	0.04
$n = 40$	100	124.80	125.00	0.02	0.09
$n = 40$	150	124.96	—	0.03	—
$n = 40$	200	124.96	—	0.04	—
$n = 40$	250	125.00	—	0.05	—
$n = 50$	100	122.00	124.10	0.03	0.14
$n = 50$	200	124.12	124.80	0.07	0.28
$n = 50$	300	124.32	124.96	0.10	0.42
$n = 50$	400	124.72	125.00	0.14	0.55
$n = 50$	800	124.80	—	0.27	—
$n = 50$	1200	124.92	—	0.41	—
$n = 50$	1600	124.96	—	0.54	—
$n = 100$[a]	100	108.32	122.68	0.14	0.58
$n = 100$[a]	200	114.76	124.32	0.28	1.16
$n = 100$[a]	300	118.00	124.88	0.42	1.74
$n = 100$[a]	400	120.44	124.96	0.56	2.23
$n = 100$[a]	500	121.56	125.00	0.70	2.90
$n = 100$[a]	1000	123.36	—	1.40	—
$n = 100$[a]	1500	123.48	—	2.11	—
$n = 100$[a]	2000	123.92	—	2.62	—
$n = 100$[a]	2500	124.00	—	3.63	—

ND = number of descents; NOPT = number of optima (out of 125); T_{avg} = average CPU time (s).
[a]Only known best upper bounds are available for $n = 100$.

randomly generated instances are given for each size n=40, 50, 100. The algorithm is implemented in C++ and runs under the Windows 2000 operating system on a HP "Kayak" 800 MHz Personal Computer.

In Table 1 we compare the SWAP-based procedure of [1] (kindly provided by the authors) and the proposed GPI-based algorithm. The column *ND* reports the number of descents (to a local optimum) performed. All the values are averaged over 25 independent runs. The average deviation from optimum is not reported since it is always negligible (in no test it became larger than 0.005%).

For each class of instances and for equal values of ND, GPI strongly outperforms SWAP in terms of solution quality because of its wider neighborhood. The improvement is particularly relevant for the $n = 100$ instances. We note that, even if GPI requires on average a larger CPU time than SWAP (about four times slower), for equivalent computation time the quality of the solutions improves significantly, making the GPI neighborhood the best suited for large instances. For equivalent CPU time, the GPI algorithm still dominates SWAP on $n = 50$, while it is only slightly dominated on the small $n = 40$ instances, where however it reaches all the optimal values within 0.09 seconds on the average.

We tried to improve the best-known upper bounds found by Congram et al. [1] for the $n = 100$ instances by running ten independent tests with $ND = 10,000$, but no better values were found.

Table 2 compares iterated GPI-based dynasearch with the ant-colony optimization (ACO) algorithm proposed in [6], where a GPI neighborhood structure is applied in the ACO framework. Following [6], the comparison is made on the time T_{opt} needed to find the optimal value. Note that the CPU times of [6] should be scaled by a factor 0.56 since those tests ran on a slower processor. The iterated dynasearch algorithm

Table 2
Comparison between GPI-based dynasearch (GPI-DS) and ACO algorithm in terms of CPU time

		$T_{\text{opt,min}}$	$T_{\text{opt,avg}}$	$T_{\text{opt,max}}$
$n = 40$	GPI-DS[a]	0.001	0.003	0.125
$n = 40$	ACO[b]	0.004	0.088	1.720
$n = 50$	GPI-DS[a]	0.001	0.010	0.562
$n = 50$	ACO[b]	0.006	0.320	10.740
$n = 100$	GPI-DS[a]	0.001	0.107	3.907
$n = 100$	ACO[b]	0.018	6.990	86.260

T_{opt} = time to find the optimal value (s).
[a]On a 800 MHz PC.
[b]On a 450 MHz PC.

reaches the same ACO solution quality within significantly smaller CPU times, even taking into account such scaling.

References

[1] R.K. Congram, C.N. Potts, S. Van de Velde, An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, INFORMS J. Comput. 14 (2002) 52–67.
[2] F. Della Croce, Generalized pairwise interchanges and machine scheduling, European J. Oper. Res. 83 (1995) 310–319.
[3] E.L. Lawler, A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness, Ann. Discrete Math. 1 (1977) 331–342.
[4] C.N. Potts, S. Van de Velde, Dynasearch—iterative local improvement by dynamic programming: part I, the traveling salesman problem, Technical Report, University of Twente, The Netherlands, 1995.
[5] A.H.G. Rinnooy Kan, B.J. Lageweg, J.K. Lenstra, Minimizing total costs in one-machine scheduling, Oper. Res. 23 (1975) 908–927.
[6] T. Stützle, M. Den Besten, M. Dorigo, Ant Colony Optimization for the total weighted tardiness problem, Technical Report IRIDIA/99-16, Université Libre de Bruxelles, Belgium, 1999.