A GPI dynasearch neighborhood for the total weighted tardiness problem

F. Della Croce, A. Grosso, R. Tadei
Dipartimento di Automatica e Informatica, Politecnico di Torino
C.so Duca degli Abruzzi 24, 10124 TORINO

Abstract

Based on the work by Congram, Potts and Van de Velde [1], we develop a dynasearch neighborhood based on 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 [1].

1. Introduction

In the single machine total weighted tardiness problem denoted $1||\sum w_i T_i$ a set of jobs $1,2,\ldots i,\ldots n$ with processing times p_i, due dates d_i and weights w_i is given. The problem calls for finding a sequence $\overline{S}$ and related completion times C_i that minimizes $T(\overline{S}) = \sum_{i=1}^{n} w_i T_i$, with $T_i = (C_i - d_i)^+ = \max(C_i - d_i, 0)$.

The $1||\sum w_i T_i$ problem is NP-hard in the strong sense [3] and even the current state of the art exact algorithms suffer from relatively poor performances. 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 by Congram et al. [1] whose main feature is the ability of searching exponential size neighborhoods in polynomial time by exploiting problem structure. To the authors' knowledge, the implementation of the so-called *iterated* dynasearch algorithm for the $1||\sum w_i T_i$ 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 extraction and reinsertion, and have been shown to produce an effective neighborhood for several scheduling problems [2].

In this note we develop a GPI-based dynasearch neighborhood which widens the swap-based one of [1]. We utilize elimination rules to efficiently search this wider neighborhood and show that it allows to achieve a better solution quality with a small CPU time increase.

2. The GPI dynasearch neighborhood

Consider a job sequence $S = (1,2,\ldots,n)$ and a pair of indices $k < l$, so that $S = \sigma k \pi l \omega$. Accordingly with [2], we define four job interchange operators API (Adjacent Pairwise Interchange) NAPI (Non Adjacent Pairwise Interchange), EBSR (Extraction and

Backward Shifted reinsertion) and EFSR (Extraction and Forward Shifted Reinsertion) which act on the sequence as follows:

$$API(k,l): \sigma k \pi l \omega \Rightarrow \sigma l \pi k \omega, \quad (\pi = \varnothing);$$
$$NAPI(k,l): \sigma k \pi l \omega \Rightarrow \sigma l \pi k \omega, \quad (\pi \neq \varnothing);$$
$$EBSR(k,l): \sigma k \pi l \omega \Rightarrow \sigma \pi k l \omega, \quad (\pi \neq \varnothing);$$
$$EFSR(k,l): \sigma k \pi l \omega \Rightarrow \sigma l k \pi \omega. \quad (\pi \neq \varnothing).$$

API and NAPI operators are grouped together in the so-called SWAP operator applied in [1]. Two job interchanges 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 sequence 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 sequences to be searched in $O(n^3)$ time by means of a dynamic programming recursion.

We develop a GPI-based dynasearch neighborhood as follows. Given the initial sequence *S* as above, at each stage *i* of the recursion, we consider the sequence $\sigma_i = ([1],[2],\ldots,[i])$ defined as the sequence with minimum tardiness among those reachable from $(1,\ldots,i)$ through a series of independent interchanges. Let *f(i)* be the total weighted tardiness of σ_i. Then, for the GPI operators the dynasearch recursion can be stated, for indices $j = 0,1,\ldots,n,$ as follows:

$$f(0) = 0,$$
$$f(j) = \min\left\{(C_{j-1} + p_j - d_j)^+, \min_{1 \le i \le j-2}\left\{f(i) + I^{\gamma}(i+1, j)\right\}\right\},$$

where $I^{\gamma}(i+1, j)$ is the weighted tardiness of the partial sequence $(i+1,\ldots, j)$ under application of the considered GPI operator. $I^{\gamma}(i+1, j)$ is set as follows.

$$I^{\gamma}(i+1,j) = \begin{cases} w_{i+1}(C_j - d_{i+1})^+ + \sum_{l=i+2}^{j}(C_l - p_{i+1} + p_j - d_l)^+, & if\ \gamma = SWAP \\ w_{i+1}(C_j - d_{i+1})^+ + \sum_{l=i+2}^{j}(C_l - p_{i+1} - d_l)^+, & if\ \gamma = EBSR \\ w_j(C_{i+1} - p_{i+1} + p_j - d_j)^+ + \sum_{l=i+1}^{j-1}(C_l + p_j - d_l)^+, & if\ \gamma = EFSR \end{cases}$$

We evaluate each $f(j)$ for γ = SWAP, EBSR, EFSR The best series of interchanges is found for $j = n$. If only the operator γ = SWAP is considered, the recursion of Congram et al. [1] is obtained. Evaluating a single API, NAPI, EBSR or EFSR requires at most $O(n)$ time. No more than j pairs of jobs 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. In order to speed up the procedure, the evaluation of a job interchange can be stopped as soon as it proves to be nonoptimal. Let $k = i+1$. We use the following

Elimination Rules The NAPI interchange between k and j needs not to be considered if

(a) $p_j > p_k, w_j < w_k$ and $\max(C_k - p_k + p_j, d_j) > d_k$, or

(b) $w_k \geq w_j, d_k \leq d_j$ and $d_j + p_j \geq C_j$, or.

(c) $C_j \leq d_j$.

Proof (sketch) The proof rests on known results about the $1||\sum w_i T_i$ problem. The NAPI interchange is proven to be dominated (and can hence be discarded) if we can find a different series of independent GPI leading to a job sequence with lower tardiness. At stage j, consider any sequence $S = \sigma k \pi j \omega$. It is known (see [4]) that condition (a) implies $T(\sigma k \pi j \omega) \leq T(\sigma j \pi k \omega)$, hence the corresponding NAPI is dominated. Similarly, condition (b) implies $T(\sigma \pi k j \omega) \leq T(\sigma j \pi k \omega)$, the latter sequence being obtained by applying EBSR between $k = i+1$ and j-1. Hence the considered NAPI is dominated. A similar argument holds for (c).

Note that Rule 1(b) cannot be utilized in the SWAP neighborhood of [1].

3. Computational results

In order to assess the behavior of the proposed neighborhood, we set it in the same iterated dynasearch algorithm developed in [1], to which we refer for a detailed description. We also used the same test instances drawn from the OR-library. The algorithm was implemented in C++ and ran under the Windows 2000 operating system on a HP "Kayak" XU-800 Personal Computer. Two versions of the code were created, to test both a SWAP dynasearch neighborhood and a GPI-dynasearch one.

In Table 1 we compare the SWAP version and the GPI version of the algorithm. The column NI reports the number of descents performed. All the values are averaged over ten independent runs. In all cases the GPI algorithm reaches better performances than the SWAP one, exhibiting only a cheap increase of the average CPU time. Note that for n = 100 "optimality" can be checked only against the best known upper bounds.

Admittedly, our implementation of the SWAP-based dynasearch algorithm performs slightly worse than that of [1], where NOPT = 125.0 is reached with NI = 150, 900 for n = 40 and 50 respectively. For n = 100, NOPT = 107.2, 116.6, and 120.9 with NI = 100, 200, 500 respectively. However, the GPI algorithm significantly improves also with respect to those results.

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

	NI	$E_{\%,avg}$		NOPT		T_{avg}	
		SWAP	GPI	SWAP	GPI	SWAP	GPI
n=40	50	0.0478	0.0009	119.3	124.3	0.03	0.06
n=40	100	0.0151	0.0000	123.0	125.0	0.06	0.11
n=40	150	0.0022	0.0000	124.3	125.0	0.10	0.17
n=40	200	0.0009	0.0000	124.6	125.0	0.13	0.23
n=50	100	0.0258	0.0012	113.0	124.0	0.12	0.20
n=50	150	0.0198	0.0006	116.8	124.6	0.16	0.29
n=50	200	0.0149	0.0005	119.0	124.7	0.23	0.39
n=50	500	0.0059	0.0002	122.2	124.9	0.57	0.95
n=50	600	0.0049	0.0000	122.9	125.0	0.68	1.16
n=50	900	0.0034	0.0000	123.8	125.0	1.01	1.72
n=100	100	0.0291	0.0036	93.3	122.5	0.79	1.02
n=100	200	0.0128	0.0001	103.8	124.4	1.56	1.99
n=100	500	0.0032	0.0000	115.7	124.9	3.94	5.04
n=100	600	0.0025	0.0000	117.2	125.0	4.79	6.07

NI = number of descents; NOPT = number of optima (out of 125);
E %,avg = average deviation from optimum; Tavg = average CPU time (s).

4. Conclusion

In this work it is shown out that a dynasearch neighborhood for the $1||\sum w_i T_i$ problem can be profitably extended to a larger class of job interchange operators. Despite of the wider neighborhood to be considered, a more efficient search procedure is obtained and improved computational results are provided for a consolidated set of instances. We believe that the extension to the GPI operators is worth to be considered for any dynasearch based algorithm for sequencing problems.

References

[1] Congram, R. K., Potts, C. N., Van de Velde, S., "An iterated dynasearch algorithm for the single machine total weighted tardiness problem", *INFORMS Journal on Computing*, to appear.

[2] Della Croce, F. "Generalized Pairwise Interchanges and machine scheduling", *European Journal of Operational Research*, **83**, 310–319 (1995).

[3] Lawler, E. L., "A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness", *Annals of Discrete Mathematics*, **1**, 331–342 (1977).

[4] Rinnooy Kan, A. H. G., Lageweg, J., Lenstra, J. K. "Minimizing total costs in one-machine scheduling", *Operations Research*, **23**, 908–927 (1975).