Computers & Industrial Engineering 98 (2016) 1–10

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Breakout dynasearch for the single-machine total weighted tardiness problem

Junwen Ding [a], Zhipeng Lü [a,b,*], T.C.E. Cheng [b], Liping Xu [a]

[a] SMART, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, PR China
[b] Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Kowloon, Hong Kong, China

ARTICLE INFO

Article history:
Received 29 September 2015
Received in revised form 8 March 2016
Accepted 28 April 2016
Available online 7 May 2016

Keywords:
Heuristic
Single-machine scheduling
Breakout local search
Adaptive perturbation strategy
Dynasearch

ABSTRACT

We present a breakout dynasearch algorithm (BDS) for solving the single-machine total weighted tardiness problem, in which a set of independent jobs with distinct processing times, weights, and due dates are to be scheduled on a single machine to minimize the sum of the weighted tardiness of all the jobs. BDS explores the search space by combining the dynasearch procedure and the adaptive perturbation strategy. Experimental results show that BDS virtually solves all the standard benchmark problem instances with 40, 50, and 100 jobs from the literature within 0.1 s. For 500 larger instances with 150, 200, 250, and 300 jobs, BDS obtains all the upper bounds with the same objective function of the optimal solutions within an average of 252 s, demonstrating the efficacy of BDS in terms of both solution quality and computational efficiency. We also analyze some key features of BDS to identify its success factors.

1. Introduction

We consider the single-machine total weighted tardiness problem (SMTWT), denoted as $1||\sum w_iT_i$. In this problem, a set of independent jobs ($N = \{1, \ldots, n\}$) is to be processed without interruption on a single machine that can process only one job at a time. Each job $i \in N$ has an integer processing time p_i, a due date d_i, and a positive weight w_i. All the jobs are available for processing at time zero. Given a job sequence, the completion time C_i and tardiness $T_i = max\{0, C_i - d_i\}$ can be calculated for each job $i \in N$. If a job i is finished after its due date, then a weighted tardiness penalty w_iT_i will incur. The objective is to find a job sequence S to minimize the sum of the weighted tardiness penalties: $f(S) = \sum_{i=1}^{n} w_iT_i$. SMTWT is an NP-hard problem (Lenstra, Kan, & Brucker, 1977), which includes many machine scheduling problems in the parallel-machines (Cheng, Hsu, & Yang, 2011), flow shops (Levner, Kats, de Pablo, & Cheng, 2010; Ng, Wang, Cheng, & Lam, 2011), job shop settings (Ji, Chen, Ge, & Cheng, 2014; Valente & Schaller, 2012).

Several exact algorithms have been proposed for solving SMTWT (Abdul-Razaq, Potts, & Van Wassenhove, 1990; Emmons, 1969; Ibaraki, 1988; Ibaraki & Nakamura, 1994; Kan, Lageweg, & Lenstra, 1975; Keha, Khowala, & Fowler, 2009; Potts & Van Wassenhove, 1985). Particularly, Pan and Shi (2007) provide an improved branch-and-bound algorithm that solves all the open OR-library benchmark instances with 100 jobs. Tanaka, Fujikuma, and Araki (2009) propose an exact algorithm based on the Successive Sublimation Dynamic Programming (SSDP) method, which can solve instances with 100 and 300 jobs within 40 s and 1 h on a 2.4 GHz Pentium IV computer, respectively. However, the algorithm needs lots of memory to store the dynamic programming states. For example, it requires 384 MB RAM to handle 300-job instances.

For large instances, exact methods are computationally inefficient to be applicable. Therefore, variants of metaheuristic algorithms have been extensively used to tackle SMTWT (Cheng, Ng, Yuan, & Liu, 2005; Congram, Potts, & van De Velde, 2002; Crauwels, Potts, & Van Wassenhove, 1998; den Besten, Stützle, & Dorigo, 2001; Sen, Sulek, & Dileepan, 2003; Volgenant & Teerhuis, 1999). Many other advanced metaheuristics have also been proposed for tackling SMTWT, such as simulated annealing (SA) (Potts & Van Wassenhove, 1991), tabu search (TS) (Bilge, Kurtulan, & Kıraç, 2007; Bożejko, Grabowski, & Wodecki, 2006), ant colony optimization (ACO) (Anghinolfi & Paolucci, 2008; den Besten, Stützle, & Dorigo, 2000; Holthaus & Rajendran, 2004), population-based variable neighbourhood search (PVNS) (Wang & Tang, 2009), particle swam optimization combined with differential evolution algorithms (PSO) (Tasgetiren, Liang, Sevkli, & Gencyilmaz, 2006), genetic algorithm (GA) (Avci, Akturk, &

* Corresponding author at: SMART, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, PR China.
E-mail addresses: dingjunwen@hust.edu.cn (J. Ding), zhipeng.lui@gmail.com (Z. Lü), lgtcheng@polyu.edu.hk (T.C.E. Cheng), xlphust@163.com (L. Xu).

http://dx.doi.org/10.1016/j.cie.2016.04.022
0360-8352/

Storer, 2003), memetic algorithm (Huang, Chang, Lim, & Zhang, 2012), hybrid approach (Wang & Tang, 2008), and variable neighbourhood descent (VND) (Geiger, 2010b, 2010a). Notably, Grosso, Della Croce, and Tadei (2004) improve the algorithms by adopting generalized pairwise interchange operators. Ergun and Orlin (2006) present a fast neighbourhood search, which improves the time complexity of searching the swap neighbourhood from $O(n^3)$ to $O(n^2)$.

We present for the first time a breakout dynasearch algorithm (BDS) based on the merits of the breakout local search (BLS) and dynasearch to tackle SMTWT. BLS has recently been shown to be effective in solving several combinatorial optimization problems, such as the sum colouring (Benlic & Hao, 2012), maximum clique (Benlic & Hao, 2013a), max-cut (Benlic & Hao, 2013b), quadratic assignment (Benlic & Hao, 2013c), vertex separator (Benlic & Hao, 2013d), and Steiner tree problems (Fu & Hao, 2014). As an improved version of BLS, BDS incorporates the dynasearch proposed by Congram et al. (2002) as a local search procedure to improve the search intensification. Besides, by introducing an adaptive perturbation mechanism, BDS can reach a suitable degree of diversification by dynamically determining the number of perturbation moves (i.e., the jump magnitude) and by adaptively choosing between several types of perturbation moves of different intensities. This is achieved by analyzing the history information stored in a dedicated memory structure.

Computational results show that BDS can easily obtain the upper bounds with the same objective function of the optimal solutions for all the standard benchmark problem instances with 40, 50, and 100 jobs in the literature in less than 0.1 s with a hit ratio of 100%. Applied to the instances with 150, 200, 250, and 300 jobs, BDS is competitive with the exact approach in the literature in that it can obtain the upper bounds with the same objective function of the optimal solutions for all of them within an average of 252 s.

We organize the remaining part of this paper as follows: Section 2 gives the details of the proposed BDS, including the main scheme of BDS, the initialization of solutions, the dynasearch procedure, and the adaptive diversification mechanism. Section 3 reports the computational results of evaluating the performance of BDS against state-of-the-art solution methods for SMTWT in the literature. Section 4 analyzes the key features of BDS to identify its success factors, while Section 5 concludes the paper and suggests future research topics.

2. Breakout dynasearch algorithm

2.1. Main scheme

Breakout dynasearch is a general stochastic local search method that follows the basic scheme of iterated local search (ILS) (Lourenço, Martin, & Stützle, 2003). The basic idea of BDS is to use a dynasearch procedure to intensify the search in a given search region, and to apply effective perturbations to move to a new search region once a local optimum is obtained. The perturbation phase is the essential part of BDS, which uses an adaptive and multi-type perturbation mechanism to introduce a suitable degree of diversification at a certain stage of the search process for the optimal solutions.

The general architecture of BDS is described in Algorithm 1. It is composed of the following five procedures: *GenerateInitialSolution* generates an initial solution for the search; *Dynasearch* is the descent local search procedure that searches a defined neighbourhood for a solution until a local optimum is obtained; *DetermineJumpMagnitude* determines the number L of perturbation moves (also called "jump magnitude"); *DeterminePerturbationType* determines the type T of perturbation moves between two or several alternatives of different perturbation type. Once the number L and the type T of perturbation moves are selected, BDS invokes the perturbation procedure to apply L moves of type T to the best solution found so far. This perturbed solution becomes the new starting point of the search whereby a new round of *Dynasearch* is performed to optimize the objective function. The relevant information recorded in the search history is used to influence the decisions made in *DetermineJumpMagnitude*, *DeterminePerturbationType*, and *Perturbation* procedures. In the following subsections we present the main components of BDS in detail.

Algorithm 1. Pseudo-code of BDS for SMTWT

```
1: Input: Processing time, weight and due time of each job in
   an unscheduled job sequence
2: Output: The best scheduled sequence S* found so far
3: S* ← GenerateInitialSolution()
4: L ← L0
5: while stopping condition not reached do
6:    S' ← Dynasearch(S*)
7:    L ← DetermineJumpMagnitude(L, S', history)
8:    T ← DeterminePerturbationType(S', history)
9:    if f(S') < f(S*) then
10:      S* = S'
11:   end if
12:   S* ← Perturbation(L, T, S*, history)
13: end while
```

2.2. Initial solution

In order to improve search diversification, a greedy and randomized construction strategy is used to generate different initial solutions of relatively good quality for multiple runs of the procedure. Specifically, each job is inserted one by one in non-decreasing order of d_i/w_i into the partial sequence S_0 (empty at the beginning) in the position that gives the least increased objective value with probability α (if it has more than one positions, select one randomly). Otherwise, the job is inserted into a randomly selected position.

2.3. Dynasearch procedure

Dynasearch (DS) is a neighbourhood search algorithm whose main feature is the ability of searching exponential-sized neighbourhoods in polynomial time by exploiting the problem structure. In essence, DS applies a series of independent moves produced by dynamic programming, which can optimize the objective function value as far as possible, while most traditional neighbourhood search procedures only apply one best move in each iteration.

Swap, forward insert, and backward insert are the three neighbourhood structures that are commonly used in metaheuristic algorithms for scheduling problems. Computational experiments suggest that, for SMTWT, the swap neighbourhood is superior to the other two neighbourhoods in terms of solution quality when used in heuristic algorithms (Geiger, 2010b). Therefore, we only use the swap neighbourhood in the dynasearch procedure. To facilitate explanation, we adopt the following notations in BDS.

$S = (s(1), \ldots, s(n))$: a sequence that gives the processing order of all the jobs.

$swap(i,j)$: a single move that swaps job i and job j $(i,j \in N)$.