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

A R T I C L E I N F O

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

A B S T R A C T

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.

© 2016 Elsevier Ltd. All rights reserved.

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/© 2016 Elsevier Ltd. All rights reserved.

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: $S^* \leftarrow GenerateInitialSolution()$
 4: $L \leftarrow L_0$
 5: while stopping condition not reached do
 6:   $S' \leftarrow Dynasearch(S^*)$
 7:   $L \leftarrow DetermineJumpMagnitude(L, S', history)$
 8:   $T \leftarrow DeterminePerturbationType(S', history)$
 9:   if $f(S') < f(S^*)$ then
10:     $S^* = S'$
11:   end if
12:   $S^* \leftarrow Perturbation(L, T, S^*, history)$
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 ($i,j \in N$).

$\Delta_{swap(i,j)}$: the incremental objective function value caused by applying a single swap $swap(i,j)$ on a given sequence ($i,j \in N$).

S_j: a partial sequence yielding the minimum total weighted tardiness for jobs $s(1),\ldots,s(j)$ ($i,j \in N$).

$F(S_j)$: the total incremental weighted tardiness for jobs $s(1),\ldots,s(j)$ by applying a series of independent swaps (Congram et al., 2002) ($i,j \in N$).

We present the dynamic programming algorithm as follows:

$$F(S_j) = \begin{cases} 0, & j = 0,1, \\ swap(1,2)^{-}, & j = 2, \\ \min\{F(S_{j-1}), \min\limits_{1 \leqslant i \leqslant j-1} \{F(S_{i-1}) + swap(i,j)^{-}\}\}, & j = 3,\ldots,n. \end{cases} \quad (1)$$

where $swap(i,j)^{-} = \min\{\Delta_{swap(i,j)}, 0\}$. The maximum incremental objective function value of S is thus equal to $F(S_n)$ and the corresponding swaps can be found by backtracking.

Algorithm 2. Pseudo-code of the Dynasearch procedure

```
1: Input: An initial sequence $S^0$
2: Output: The local optimal sequence $S^*$ found so far
3: $S^* \leftarrow S^0$
4: $M \leftarrow$ two_swap($S^0$)
5: while dynamic_programming($M$) $< 0$ do
6:     $S^* \leftarrow$ backtracking($S^*$)
7:     $M \leftarrow$ two_swap($S^*$)
8: end while
9: return $S^*$
```

Algorithm 2 gives the pseudo-code of the dynasearch procedure. DS starts with an initial sequence S^0 and a two-dimensional matrix M that stores the incremental objective value of each swap in the sequence (i.e., $M(i,j) = \Delta_{swap(i,j)}, 1 \leqslant i < j \leqslant n$). In this paper we employ the fast neighbourhood search algorithm (Ergun & Orlin, 2006) to improve the efficiency of the original dynasearch algorithm (Congram et al., 2002), which only requires $O(n^2)$ time to search the dynasearch swap neighbourhood. The function $two\_swap(S)$ gives the incremental objective values of all the possible swaps of the corresponding sequence S. The function $dynamic\_programming(M)$ returns the maximum incremental objective value of a given sequence by employing the best set of independent swaps. This procedure terminates when the incremental value is non-negative, which means that there is no further improvement and the solution is trapped in a local optimum.

2.4. Adaptive diversification mechanism

2.4.1. Determining jump magnitude

When a local optimum S is obtained in the dynasearch phase, the local optimum $f(S)$ is stored in a hash table HT (if it is not already in HT), along with the iteration number $iter_{last\_visit}$ at which S was last visited. More precisely, the index h where $f(S)$ and $iter_{last\_visit}$ should be mapped in the hash table is calculated through the formula $h = f(S) mod[MAXHS + 1]$, where $MAXHS$ is the size of the hash table.

If there is no solution recorded at HT_h, the local optimum $f(S)$ and $iter_{last\_visit}$ are stored in HT_h. Otherwise, keep passing to the next location until an identical value is encountered or a free place is found to place $f(S)$ and $iter_{last\_visit}$.

After a local optimum S is obtained during the dynasearch phase, BDS determines a suitable number of perturbation moves for the next perturbation phase. This procedure is given in Algorithm 3 and the corresponding flow chat is presented in Fig. 1.

Algorithm 3. Pseudo-code of the DetermineJumpMagnitude

```
1: Input: Local optimum $S$ returned by DS, current jump
   magnitude $L$ and history information including the hash
   table $HT$ of previously encountered local optimum, the
   descent-phase number $lc$ when the last cycle was
   encountered, the current descent-phase number $iter_{cur}$, and
   the number $w$ of consecutive iterations of non-improving
   global optima.
2: Output: Jump magnitude $L$ for the next perturbation phase
3: $wc \leftarrow 10, num\_nc \leftarrow 1$
4: $prev\_visit \leftarrow PreviousEncounter(HT, f(S))$ /* Check
   whether $f(S)$ has previously been encountered */
5: if $f(S) < f(S^*) \| w > T_0$ then
6:     $w \leftarrow 0$
7: else
8:     $w \leftarrow w + 1$
9: end if
10: if $prev\_visit \neq -1$ then
11:     $wc \leftarrow wc + iter_{cur} - lc$ /* A cycle is encountered */
12:     $num\_nc \leftarrow num\_nc + 1$
13:     $lc \leftarrow iter_{cur}$
14:     $L \leftarrow L + 1$
15: else if $(iter_{cur} - lc) > (wc/num\_nc) \cdot \beta$ then
16:     $L \leftarrow L - 1$
17: end if
18: if $L > L_{max}$ then
19:     $L \leftarrow L_{max}$
20: else if $L < L_{min}$ then
21:     $L \leftarrow L_{min}$
22: end if
```

BDS first calls a function $PreviousEncounter$ which checks whether $f(S)$ is already in the hash table. If $f(S)$ is not in HT, the function $PreviousEncounter$ returns -1, thus $f(S)$ and the corresponding descent-phase number $iter_{cur}$ is inserted in HT. Otherwise, if S has previously been visited during the search, the record of the current descent-phase number $iter_{cur}$ with respect to the local optimum $f(S)$ in HT is updated and the function returns the iteration at which $f(S)$ has been encountered earlier. Parameter w records the number of consecutive iterations of non-improving global optima, which is used to determine the perturbation type. If the global optima is improved (i.e., $f(S) < f(S^*)$), or w exceeds a predefined threshold value T_0 (i.e., $w > T_0$), w is set to 0. Otherwise, w is increased by 1.

If a cycle is encountered, BDS increases the number of perturbation moves to slightly increase the degree of diversification. Otherwise, the number of perturbation moves is decreased if a cycle has not been detected for at least $(wc/num\_nc) \cdot \beta$ descent phases, where wc/num_nc is the average number of descent phases between two consecutive cycles and β is a coefficient. Finally, the number of perturbation moves is limited to values in the interval $[L_{min}, L_{max}]$.

2.4.2. Determining perturbation type

BDS employs both random and directed perturbations to guide the search towards new regions of the search space.

The directed perturbation is based on the idea of tabu list from tabu search given in Bilge et al. (2007). It is an attribute-based tabu strategy that contains more information about an already visited solution. In a given permutation, if job i is directly preceded by

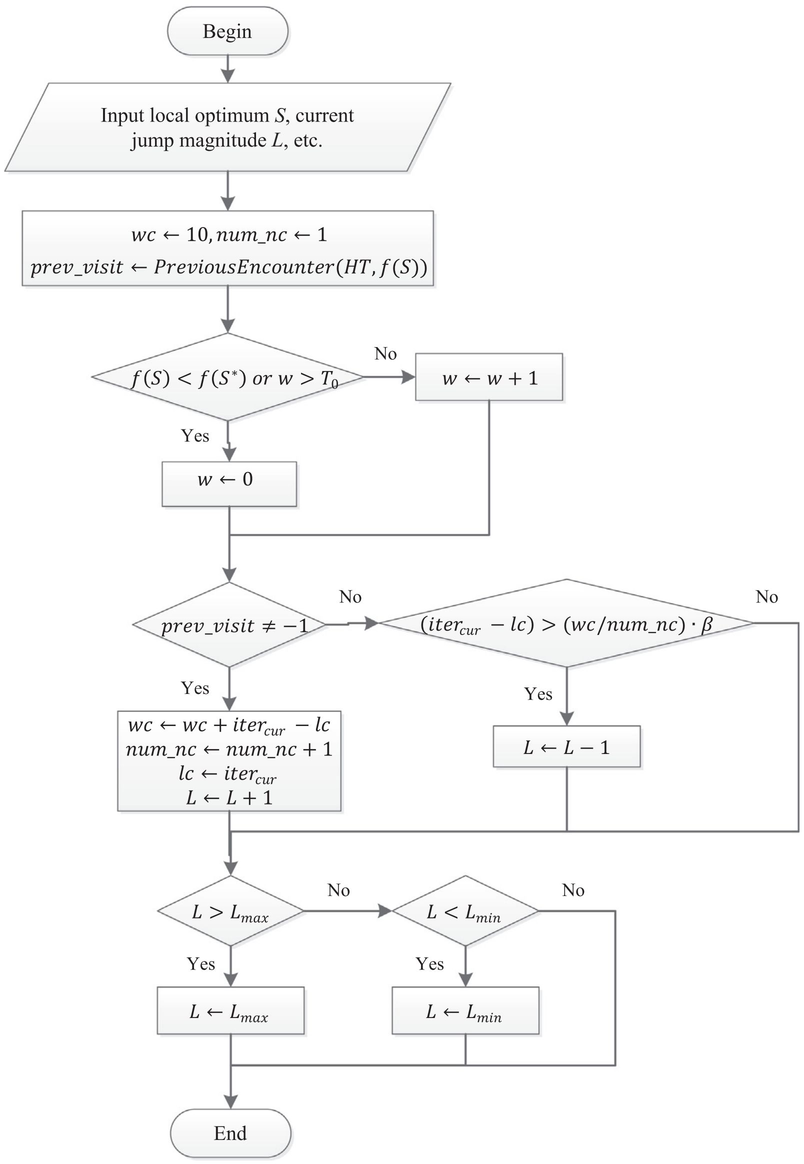
Begin
Input local optimum S, current jump magnitude L, etc.
$wc \leftarrow 10, num\_nc \leftarrow 1$
$prev\_visit \leftarrow PreviousEncounter(HT, f(S))$
$f(S) < f(S^*)$ or $w > T_0$
No
$w \leftarrow w+1$
Yes
$w \leftarrow 0$
$prev\_visit \neq -1$
No
$(iter_{cur} - lc) > (wc/num\_nc) \cdot \beta$
No
Yes
$wc \leftarrow wc + iter_{cur} - lc$
$num\_nc \leftarrow num\_nc + 1$
$lc \leftarrow iter_{cur}$
$L \leftarrow L+1$
Yes
$L \leftarrow L-1$
$L > L_{max}$
No
$L < L_{min}$
No
Yes
$L \leftarrow L_{max}$
Yes
$L \leftarrow L_{min}$
End

Fig. 1. The flow chat of Algorithm 3.

job $i-1$, it is referred to arc $((i-1), i)$. The attribute-based tabu strategy aims to prohibit some of the arcs broken by recent moves to be constructed again during the tabu tenure. Precisely, if job i and job j $(i < j)$ are swapped, four arcs that are broken, namely arcs $((i-1), i), (i, (i+1)), ((j-1), j)$, and $(j, (j+1))$, are classified as tabu, and these arcs are forbidden to be constructed again for θ iterations (tabu tenure). In the directed perturbation, only the moves that do not re-construct the arcs during the dynasearch procedure are permitted. The directed perturbation relies thus both on (1) history information that keeps track, for each move, of the last iteration when it was performed and (2) the quality of the moves to be applied for perturbation in order not to deteriorate too much the perturbed solution.

The random perturbation, which is significantly stronger than the directed perturbation, consists of two kinds of moves in this paper: forward insert and swap. In a forward insert move, two jobs i and j $(i < j)$ are randomly selected, and job i is inserted after job j. In a swap move, two randomly selected jobs are simply swapped. It is reasonable that the forward insert perturbation is much stronger than the same jump magnitude of swap perturbation, because the swap neighbourhood is employed in the dynasearch procedure, and DS may be easier to fall back to the previous local optimum after the swap perturbation.

Once a search stagnation is detected, i.e., the number w of consecutive iterations where the best found solution has not been improved exceeds a threshold value T_0 (i.e., $w > T_0$), BDS applies the forward insert perturbation in order to drive the search towards distant regions of the search space. Otherwise, BDS alternates probabilistically between directed perturbation and random perturbation (swap). The probability of applying a particular perturbation is determined by the following equation:

$$PT = \begin{cases} directed\ pert., & \text{if } p > rand(0,1); \\ random\ pert., & \text{otherwise.} \end{cases} \tag{2}$$

$$p = \begin{cases} 1 - e^{-\frac{iter_{cur} - lc}{max\_inc}}, & \text{if } prev\_visit = -1; \\ \gamma + e^{-\frac{iter_{cur} - prev\_visit}{num\_opt\_HT}}, & \text{otherwise.} \end{cases} \tag{3}$$

where $iter_{cur}$ is the current descent-phase number, lc is the number of the last descent-phase when a cycle is encountered, max_inc is the maximum number of consecutive descent phases without returning to a previous solution, num_opt_HT is the number of local optima stored in the hash table, and γ is a positive coefficient that takes a real value in the range $[0,1]$.

The logic of Eq. (3) is as follows: In the first case, when $f(S)$ has not been recently visited, i.e., $f(S)$ is not in HT and thus $prev\_visit = -1$, BDS determines the probability of applying directed perturbation over random perturbation depending on the difference $iter_{cur} - lc$, i.e., the number of consecutive descent phases elapsed without returning to a previous local optimum from HT. The larger the difference $iter_{cur} - lc$ is, the less is the chance that the search is trapped in a cycle, and Eq. (3) ensures the higher is the probability of using directed perturbation over random perturbation.

In the second case, when $f(S)$ has previously been visited, i.e., $prev\_visit \neq -1$, BDS determines the probability of using directed perturbation depending on the difference $iter_{cur} - prev\_visit$, i.e., the number of descents elapsed before a cycle is encountered. The increment of $iter_{cur} - prev\_visit$ means that the search is more likely to be trapped in a large cycle. Since large cycles are more difficult to break than small ones, Eq. (3) ensures that with the increment of $iter_{cur} - prev\_visit$, there are more chances of performing random perturbation moves, which can introduce more diversification to the search.

Note that the procedure of determining the jump magnitude and the perturbation type are highly related. The parameters used in the procedure of determining the perturbation type are the same as those in the procedure of determining the jump magnitude. For example, when an local optimum solution S is obtained by dynasearch, the number w of consecutive iterations where the best found solution has not been improved exceeds a threshold value T_0 (i.e., $w > T_0$), forward insert perturbation is adopted. Otherwise, there are two situations: First, if $f(S)$ has not been recently visited, i.e., $prev\_visit = -1$, the variable p is determined by the first formula in Eq. (3), which depends on the difference $iter_{cur} - lc$. Second, if $f(S)$ has not been previously visited, i.e., $prev\_visit \neq -1, p$ is determined by the second formula in Eq. (3), which depends on the difference $iter_{cur} - prev\_visit$. After p is determined, we generate a random number between 0 and 1, if this number does not exceed p, directed perturbation is adopted. Otherwise, random perturbation is adopted.

3. Computational results

3.1. Test instances and experimental protocol

In this section we present the results of extensive experiments conducted on two sets of test problems to evaluate the performance of BDS. The first set is the standard benchmark problem instances from the OR-library,[1] which has shown to be easy to solve (Geiger, 2009). The benchmark set consists of 375 instances of three different sizes. For each size $n = 40$, 50, and 100, a sample of 125 instances are provided. The optimal results were obtained by Pan and Shi (2007). The second set of benchmark instances comprises 500 instances with problem size $n = 150$, 200, 250, and 300. To the best of our knowledge, these instances have only been solved by Tanaka et al. (2009) with an exact algorithm. Both the instances and the optimal results can be found on the website.[2]

We programmed the proposed BDS algorithm in C and ran it on a computer with an Intel Xeon E5440 2.83 GHz CPU and 2 GB RAM. In the following experiments, we applied BDS to each problem instance for 20 independent runs. The experimental results show that different problem sizes require different appropriate parameter settings. We have conducted preliminary experiments to find out the best values of all the parameters used in our BDS algorithm on all the tested instances with 40, 50, 100, 150, 200, 250, and 300 jobs. In addition, we have conducted a detailed analysis and adjustment of two most important parameters (β, γ) in Section 4.2. Table 1 gives the descriptions and the best settings for the main parameters to run BDS in this study.

Following past researchers, we report the following values for each instance:

(1) pd: The percentage relative deviation of a solution value f_s found by an algorithm from the optimal or best-known solution value OPT, i.e., $pd = 100(f_s - OPT)/OPT$. (When $OPT = 0, pd = 100f_s$ is alternatively used)
(2) ad: The average pd value for a sample of 125 instances in 20 trial runs.
(3) md: The maximum pd out of a sample of 125 instances in 20 trial runs.
(4) no: The number of optimal or best-known solution values found out of a sample of 125 instances.
(5) tm: The average computational time in seconds for a sample of 125 instances in 20 trial runs.
(6) n_{hit}: The average hit ratio to the optimal or best-known solution for a sample of 125 instances in 20 trial runs.

3.2. Comparison between BDS and other metaheuristics

The first experiment aims to evaluate the performance of BDS to tackle standard benchmark instances with 40–100 jobs and compare BDS with the state-of-the-art metaheuristic algorithms in the literature. We applied BDS to each instance for 20 independent runs and stopped when it obtained the optimal solution or reached the maximum run-time of one minute for each run. Table 2 summarizes the results of the experiment, where the values are indicated in bold if the corresponding algorithm obtains the reported results with the least computational time compared with other reference algorithms.

Table 2 reports the results of comparing BDS with nine recently proposed metaheuristic algorithms in the literature. The first is the iterated local search (ILS) proposed by den Besten et al. (2001). The second is the problem space genetic algorithm (PSGA) proposed by Avci et al. (2003). The third is the tabu search (TS) algorithm proposed by Bilge et al. (2007). The fourth to sixth are the variable neighbourhood search (VNS) algorithm, the particle swarm optimization (PSO) algorithm, and the differential evolution (DE) algorithm proposed by Tasgetiren et al. (2006). The seventh is the population-based variable neighbourhood search (PVNS) algorithm proposed by Wang and Tang (2009). The eighth is the variable neighbourhood descent (VND) algorithm proposed by Geiger (2010b). The ninth is the iterated enhanced dynasearch algorithm (GPI-DS) proposed by Grosso et al. (2004).

Table 2 shows that BDS is competitive with the referenced algorithms. In detail, all the algorithms, except PSGA and TS, can obtain the upper bounds with the same objective function of the optimal solutions for all the instances. VNS and VND, which are based on variable neighbourhood search, are inferior to the other algorithms in terms of solution quality due to their high values of ad. Although PSO, DE, and BDS have the highest hit ratio of 100%, BDS requires the smallest computational time among all the algorithms. The results reveal that BDS has good performance in terms of both solution quality and computational efficiency. Note that in order to make a fair comparison, we implemented and ran the effective GPI-DS algorithm proposed by Grosso et al. (2004) on the same computer as ours. We see that BDS is also comparable with this effective algorithm. Specifically, for small instances with no more than 100 jobs, BDS needs slightly less computational time than

[1] http://people.brunel.ac.uk/~mastjjb/jeb/info.html.
[2] http://turbine.kuee.kyoto-u.ac.jp/~tanaka/SiPS/SiPS.html.

Table 1
Parameter settings for BDS for different problem sizes.

Parameter	Section	Description	Value
α	2.2	Proportion of greediness and randomness	0.8
β	2.4.1	Coefficient for determining jump magnitude	2
γ	2.4.2	Coefficient for determining perturbation type	0.3
θ	2.4.2	Tabu tenure in directed perturbation	$n/4$
T_0	2.4.2	Coefficient for strong perturbation	$500 + n$
L_{min}	2.4.1	Minimum number of jump magnitude	4
L_{max}	2.4.1	Maximum number of jump magnitude	$10 + n/100$
MAXHS	2.4.1	Size of the hash table	10000

GPI-DS to obtain the optimal results, while for larger instances with 150–300 jobs, BDS outperforms GPI-DS as it takes smaller computational time (see Section 3.3).

3.3. Computational results on large instances

In order to further evaluate the performance of BDS, we performed experiments on instances with up to 300 jobs to make comparison with GPI-DS and Tanaka et al.'s exact algorithm. As before, we applied both BDS and GPI-DS to each instance for 20 independent runs and stopped when they obtained the optimal solutions or the maximum run-time reached 40 min for each run. For a fair comparison, we use the program provided by Tanaka et al. and ran it on the same computers as ours (i.e., an Intel Xeon E5440 with 2.83 GHz, 2 GB RAM). We report the comparison results in Table 3, where the values are indicated in bold if the corresponding algorithm obtains the reported results with the least computational time compared with other reference algorithms.

From Table 3, we observe that BDS is also comparable with Tanaka et al.'s exact algorithm for all the instances in terms of the average computational time. Although Tanaka et al.'s exact algorithm can obtain the optimal solutions deterministically, it requires a large amount of memory to store the dynamic programming states, which is up to 384 MB to handle instances with 300 jobs. Needing an $O(n^2)$ memory space theoretically, BDS only requires about 500 KB for 300 jobs, which is much less than the memory usage of Tanaka et al.'s exact algorithm. Besides, BDS shows slightly better performance than GPI-DS in terms of both solution quality and computational time.

4. Analysis and discussion

4.1. Theoretical similarities and differences

Intensification and diversification are two powerful drivers of the high performance metaheuristics. The perturbation mechanism is a dynamic strategy for BDS to strike a proper tradeoff between intensification and diversification. By changing the jump magnitude and selecting the most suitable perturbation after each iteration according to the historical search information, BDS is intelligent enough to both intensively explore areas of the search space with high quality solutions, and to move to unexplored areas of the search space if necessary.

The general BDS procedure inherits some features from the two well-established metaheuristics: iterated local search and reactive tabu search. Besides, the effective dynasearch procedure is also incorporated into the BDS as the local search procedure. We discuss the similarities and differences between BDS and the other two algorithms in the following.

First, although BDS follows the framework of iterated local search, it distinguishes from ILS by introducing an adaptive breakout perturbation strategy to escape from local optima. Second, the best solution found so far is always accepted as the new starting solution for BDS, while in ILS the new starting solution is alternatively selected from the best found solution and the last local optimal solution by the acceptance criterion. Third, the diversification mechanism of reactive tabu search is achieved by adaptively changing the tabu tenure based on the history information, while BDS mainly focuses on the perturbation phase and keeps the tabu tenure unchanged. Finally, the directed perturbation of BDS is based on the tabu list borrowed from tabu search. However, BDS does not consider the tabu list during its dynasearch phase while each iteration of tabu search is constrained by the tabu list.

Since the iterated dynasearch algorithm (IDS) (Congram et al., 2002), the iterated enhanced dynasearch algorithm (GPI-DS) (Grosso et al., 2004), and breakout dynasearch algorithm (BDS) are all proposed to tackle the SMTWT problem, the similarities among them are: First, they all follow the basic scheme of iterated local search. Second, they all use the dynasearch procedure to optimize solutions. However, there are obvious differences: First, both swap neighbourhood and insert neighbourhood are considered in GPI-DS, while only swap neighbourhood is explored in IDS and BDS. Second, Based on some relevant information on search history, BDS introduces an adaptive perturbation mechanism which can reach a suitable degree of diversification by dynamically determining the number of perturbation moves and adaptively choosing types of perturbation moves of difference intensities, while both IDS and GPI-DS only employ a random perturbation (i.e., predefined type of random move) when the search traps into local optima.

4.2. Impact of parameters β and γ

In this section, we present the preliminary experiments to analyze the influence of the parameters β and γ on the performance of BDS. For parameter β, we take 10 different values $\beta \in [0.5, 5]$ (step size of 0.5) and keep other parameters fixed. For parameter γ, we take 11 different values $\gamma \in [0, 1]$ (step size of 0.1) and keep other

Table 2
Comparison between BDS and other metaheuristic algorithms on instances with 40, 50, and 100 jobs.

Algor.	$n = 40$					$n = 50$					$n = 100$					Run environment
	ad	*md*	n_{hit}	*tm*	*no*	*ad*	*md*	n_{hit}	*tm*	*no*	*ad*	*md*	n_{hit}	*tm*	*no*	
ILS	0.13	–	–	0.040	125	0.86	–	–	0.20	125	14.50	–	–	5.75	125	Pent. III, 700 MHz
PSGA	0.000	0.000	–	29.11	125	0.000	0.020	–	41.02	124	0.020	0.300	–	118.97	83	Pent. II, 400 MHz
TS	0.000	–	–	2.74	125	0.001	–	–	16.91	124	0.007	–	–	127.59	108	Pent. IV, 1.6 GHz
VNS	0.21	–	0.95	0.18	125	0.20	–	0.91	0.20	125	0.11	–	0.90	5.72	125	Pent. IV, 2.6 GHz
PSO	0.000	0.000	1	0.21	125	0.000	0.000	1	0.54	125	0.000	0.000	1	8.52	125	Pent. IV, 2.6 GHz
DE	0.000	0.000	1	0.20	125	0.000	0.000	1	0.54	125	0.000	0.000	1	8.70	125	Pent. IV, 2.6 GHz
PVNS	0.000	0.000	0.9971	6.19	125	0.000	0.000	0.9965	12.15	125	0.000	0.000	–	183.47	125	Pent. IV, 3.0 GHz
VND	0.99	–	–	0.442	125	1.45	–	–	1.179	125	0.98	–	–	28.527	125	Inte. X., 2.67 GHz
GPI-DS	0.000	0.000	1	0.003	125	0.000	0.000	1	0.009	125	0.000	0.000	1	0.069	125	Pent. IV, 2.88 GHz
BDS	0.000	0.000	1	**0.003**	125	0.000	0.000	1	**0.008**	125	0.000	0.000	1	**0.058**	125	Pent. IV, 2.88 GHz

Table 3
Comparison among BDS, GPI-DS, and Tanaka et al.'s exact algorithm on instances with up to 300 jobs.

n	Tanaka		GPI-DS					BDS				
	tm	*no*	*ad*	*md*	n_{hit}	*tm*	*no*	*ad*	*md*	n_{hit}	*tm*	*no*
40	0.12	125	0.000	0.000	1	0.003	125	0.000	0.000	1	**0.003**	125
50	0.35	125	0.000	0.000	1	0.009	125	0.000	0.000	1	**0.008**	125
100	7.67	125	0.000	0.000	1	0.069	125	0.000	0.000	1	**0.058**	125
150	23.38	125	0.001	0.031	0.9553	14.58	125	0.000	0.047	0.9897	**8.77**	125
200	60.46	125	0.068	0.867	0.9064	158.38	124	0.032	0.885	0.9281	**54.39**	125
250	185.72	125	0.087	0.753	0.8651	966.58	120	0.004	0.438	0.9047	**128.84**	125
300	327.69	125	0.102	0.945	0.8036	1704.37	116	0.017	0.992	0.8766	**251.06**	125

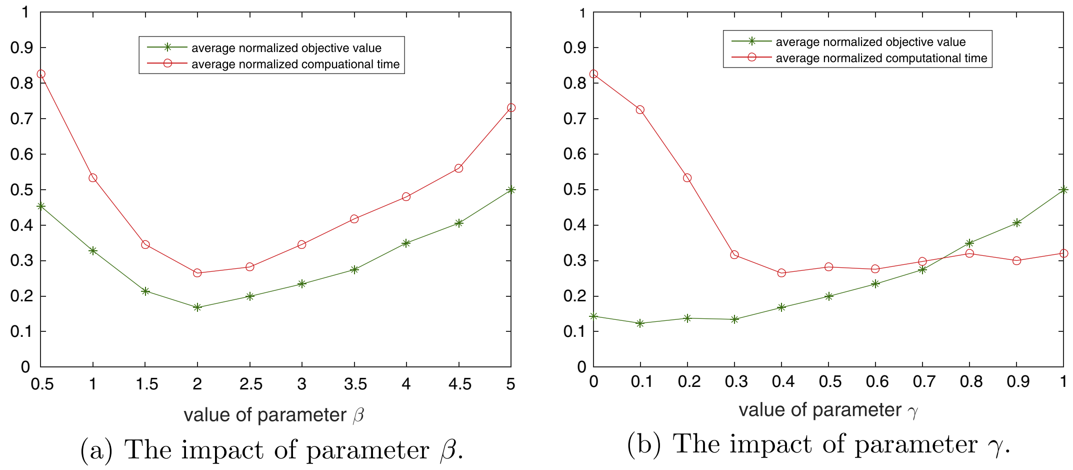

Fig. 2. The average normalized objective function value and computational time corresponding to different values of parameter β and γ.

parameters fixed. We perform 20 independent runs for each parameter on all of the tested instances with 40, 50, 100, 150, 200, 250, and 300 jobs, and stop our algorithm when it reaches $2000 + 30 * n$ iterations. Fig. 2a and b show the average normalized objective value[3] and the corresponding average normalized computational time, respectively.

From Fig. 2a, one observes that, for parameter β, both the average normalized objective value and the corresponding average computational time dramatically decreases from 0.5 to 2 and gradually increase when $\beta > 2$. This implies that too small or too large β, i.e., too quick or too slow decrement of jump magnitude L, cannot equip BDS with a proper portion of diversification. For parameter γ in Fig. 2b, one observes that the average normalized objective value is relatively small and almost remains unchanged when $\gamma \in [0, 0.3]$, and gradually increase when $\gamma > 0.3$, while the corresponding average computational time rapidly decreases when $\gamma < 0.4$ and slightly increases when $\gamma > 0.4$. This illustrates that large proportion of directed perturbation (relatively weak perturbation type) may save computational time but deteriorate the solution quality. Considering both solution quality and computational time, β and γ are suggested to be 2 and 0.3, respectively.

4.3. Comparison between different versions of BDS

In this section, we discuss the merits of the adaptive and multi-type perturbation mechanism of BDS by comparing BDS with three versions of iterated dynasearch (denoted as IDS-I, IDS-II, and IDS-III). These three IDS algorithms are constructed with slight modifications of BDS. IDS-I is constructed by fixing the probability of applying directed perturbation versus random perturbation to 0.95. IDS-II is constructed by fixing the jump magnitude ($L = 6 + n/100$). IDS-III is the most basic version of IDS that applies a fixed number ($L = 6 + n/100$) of random perturbation. We applied these four algorithms to instance wt200_111 (which has been tested to be relatively hard among instances with 200 jobs) with the same initial solutions for 20 independent runs and report the results in Fig. 3.

Fig. 3 shows how the best solutions found by the four algorithms evolve with search iterations. We see that the curve of BDS decreases more dramatically and smoothly than the other three algorithms. It reaches the optimal solutions with the least computational time. For other three algorithms, without the adaptive diversification mechanism, their curves usually decrease sharply during the earlier iterations while they remain unchanged in the later iterations, which means they are more likely to be trapped into local optima and less capable of jumping out of them and moving to unexplored regions of the search space. This demonstrates the effectiveness of the adaptive diversification mechanism embedded in BDS.

In order to further explore the behaviour of BDS, we recorded the current obtained solutions after each iteration of BDS and compared it with those of the other three algorithms on instance wt200_111. Fig. 4 depicts the results, which shows the current solutions of the four algorithms from the 200th iteration to the 800th iteration. We see that BDS behaves most reasonably: after a consecutive number of small jumps that cannot improve the best solution found so far or encounter the previous local optima, it makes several big jumps in order to get out of the local optima. The jump magnitude in IDS-I is not as regular as that in BDS, where after a small jump, it often performs a consecutive number of very

[3] Average normalized objective value: First, find the maximum and minimum objective value (denoted by f_{max} and f_{min}) over the 20 runs for each instance. Second, normalize the objective value of each run for each instance using $norm.f = (f - f_{min})/(f_{max} - f_{min} + 1)$. Third, average the $norm.f$ for all the test instances for each value of parameter β or γ.

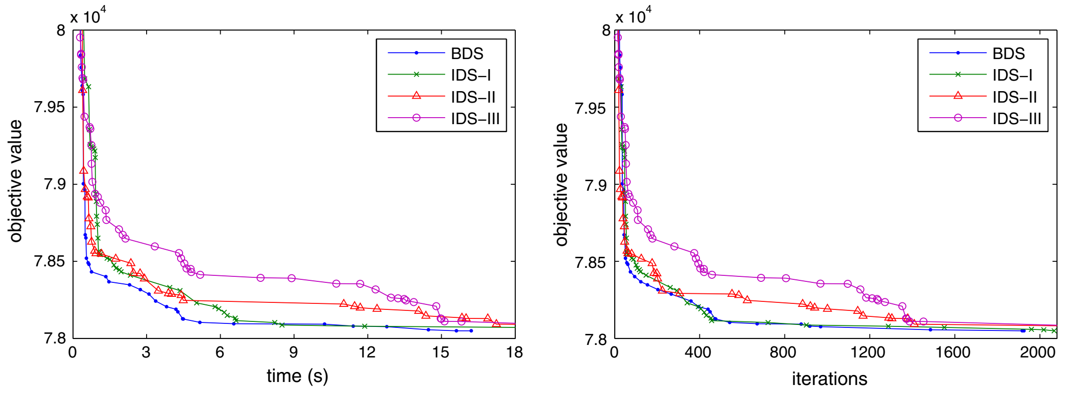

Fig. 3. Comparison between different versions of BDS in terms of the best solution found so far.

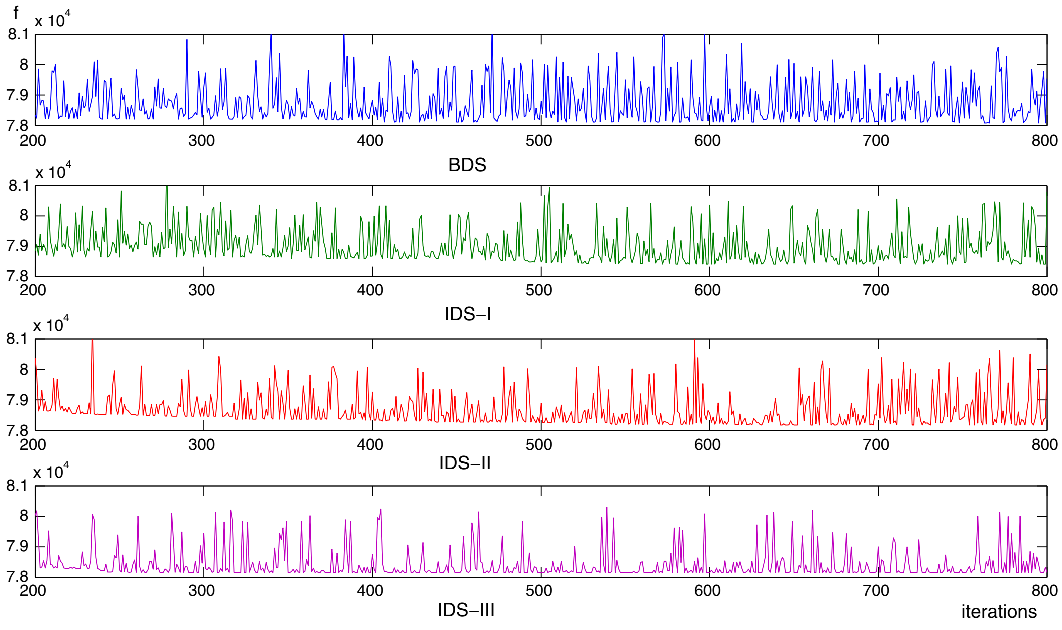

Fig. 4. Comparison between different versions of BDS in terms of current solution after each iteration.

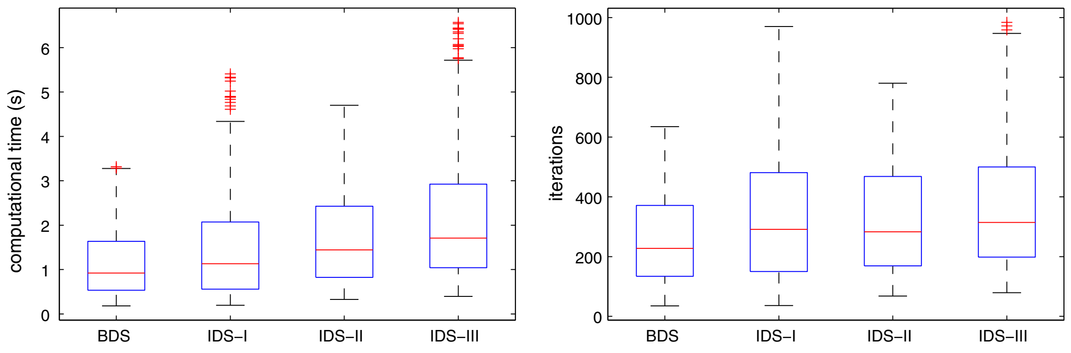

Fig. 5. Box and whisker plots corresponding to different versions of BDS on instance wt200_60 in terms of computational time and iterations in which the optimal results are obtained.

large jumps. Similarly, IDS-II is not reasonable either for it often makes several big jumps after a large number of small jumps. IDS-III is unable to get out of the local optima because it applies a fixed number of random perturbations. The results further manifest the behaviour and the efficacy of the adaptive diversification mechanism used in BDS.

Finally, we analyze the stability of BDS. To this end, we applied the four algorithms to instance wt200_60 (which has been tested to be relatively hard among the instances with 200 jobs) with 500 independent runs and stopped when an optimal solution was obtained. Fig. 5 shows the statistical results of the computational time and number of iterations at which the optimal results are obtained. The horizontal axis indicates the four tested algorithms and the vertical axis shows the performance (computational time in left sub-figure and number of iterations in right sub-figure). From Fig. 5, we observe that the length of both the computational time and the number of iterations boxes of BDS are relatively shorter than those of the other three algorithms. This indicates the good stability of BDS.

5. Conclusions

We present the breakout dynasearch algorithm for solving SMTWT. The gist of BDS is that it alternates between the dyansearch procedure and an adaptive combination of multi-type perturbations procedure to achieve a desired balance between intensification and diversification. The adaptive diversification phase is vitally important to the performance of BDS since DS alone is less effective in jumping out of local optima. The diversification mechanism of BDS is achieved by varying the jump magnitude and selecting the most suitable type of perturbation after each iteration upon obtaining a local optimum.

Applied to tackle benchmark problem instances with 40–100 jobs in the literature, BDS can obtain all the upper bounds with the same objective function of the optimal solutions in less than 0.1 s and attains a hit ratio of 100%, revealing its good performance in terms of both solution quality and computational efficiency. For large instances with 150, 200, 250, and 300 jobs in the literature, BDS can also obtain all the upper bounds with the same objective function of the optimal solutions within an average computational time of 252 s and attain an average hit ratio of 87.66%.

We analyze some key features of BDS in order to identify its success factors. From the analysis, we conclude that the good performance of BDS stems from the diversification degrees introduced in the algorithm. Finally, given the merits of the adaptive diversification mechanism of BDS, future research will focus on the following aspects: First, BDS can be combined with some nature-inspired mechanisms such as simulated annealing and ant colony optimization. Second, it is valuable to adopt self-adaptive strategy to adjusting the parameters by considering advanced artificial intelligence techniques, such as reinforced learning. Third, we are eager to extend BDS to be a general algorithmic framework to tackle other intractable combinatorial optimization problems.

Acknowledgements

The research was supported in part by the National Natural Science Foundation of China under grant numbers 61370183 and the programme for New Century Excellent Talents in University (NCET 2013). T.C.E. Cheng was supported in part by The Hong Kong Polytechnic University under the Fung Yiu King - Wing Hang Bank Endowed Professorship in Business Administration.

References

Abdul-Razaq, T., Potts, C. N., & Van Wassenhove, L. N. (1990). A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics, 26*(2), 235–253.
Anghinolfi, D., & Paolucci, M. (2008). A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem. *International Journal of Operations Research, 5*(1), 1–17.
Avci, S., Akturk, M. S., & Storer, R. H. (2003). A problem space algorithm for single machine weighted tardiness problems. *IIE Transactions, 35*(5), 479–486.
Benlic, U., & Hao, J. K. (2012). A study of breakout local search for the minimum sum coloring problem. In *Simulated evolution and learning* (pp. 128–137). Springer.
Benlic, U., & Hao, J. K. (2013a). Breakout local search for maximum clique problems. *Computers & Operations Research, 40*(1), 192–206.
Benlic, U., & Hao, J. K. (2013b). Breakout local search for the max-cut problem. *Engineering Applications of Artificial Intelligence, 26*(3), 1162–1173.
Benlic, U., & Hao, J. K. (2013c). Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation, 219*(9), 4800–4815.
Benlic, U., & Hao, J. K. (2013d). Breakout local search for the vertex separator problem. In *Proceedings of the twenty-third international joint conference on artificial intelligence* (pp. 461–467). AAAI Press.
Bilge, Ü., Kurtulan, M., & Kıraç, F. (2007). A tabu search algorithm for the single machine total weighted tardiness problem. *European Journal of Operational Research, 176*(3), 1423–1435.
Bożejko, W., Grabowski, J., & Wodecki, M. (2006). Block approach tabu search algorithm for single machine total weighted tardiness problem. *Computers & Industrial Engineering, 50*(1), 1–14.
Cheng, T. C. E., Hsu, C. J., & Yang, D. L. (2011). Unrelated parallel-machine scheduling with deteriorating maintenance activities. *Computers & Industrial Engineering, 60*(4), 602–605.
Cheng, T. C. E., Ng, C., Yuan, J., & Liu, Z. (2005). Single machine scheduling to minimize total weighted tardiness. *European Journal of Operational Research, 165* (2), 423–443.
Congram, R. K., Potts, C. N., & van De Velde, S. L. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing, 14*(1), 52–67.
Crauwels, H., Potts, C. N., & Van Wassenhove, L. N. (1998). Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing, 10*(3), 341–350.
den Besten, M., Stützle, T., & Dorigo, M. (2000). Ant colony optimization for the total weighted tardiness problem. In *Parallel Problem Solving from Nature PPSN VI* (pp. 611–620). Springer.
den Besten, M., Stützle, T., & Dorigo, M. (2001). Design of iterated local search algorithms. In *Applications of evolutionary computing* (pp. 441–451). Springer.
Emmons, H. (1969). One-machine sequencing to minimize certain functions of job tardiness. *Operations Research, 17*(4), 701–715.
Ergun, Ö., & Orlin, J. B. (2006). Fast neighborhood search for the single machine total weighted tardiness problem. *Operations Research Letters, 34*(1), 41–45.
Fu, Z. H., & Hao, J. K. (2014). Breakout local search for the steiner tree problem with revenue, budget and hop constraints. *European Journal of Operational Research, 232*(1), 209–220.
Geiger, M. J. (2009). The single machine total weighted tardiness problem – Is it (for metaheuristics) a solved problem? In *Proceedings of the 8th Metaheuristics International Conference MIC 2009* (pp. 141.1–141.10).
Geiger, M. J. (2010a). New instances for the single machine total weighted tardiness problem. Research Report.
Geiger, M. J. (2010b). On heuristic search for the single machine total weighted tardiness problem – Some theoretical insights and their empirical verification. *European Journal of Operational Research, 207*(3), 1235–1243.
Grosso, A., Della Croce, F., & Tadei, R. (2004). An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Operations Research Letters, 32*(1), 68–72.
Holthaus, O., & Rajendran, C. (2004). A fast ant-colony algorithm for single-machine scheduling to minimize the sum of weighted tardiness of jobs. *Journal of the Operational Research Society, 56*(8), 947–953.
Huang, W. H., Chang, P. C., Lim, M. H., & Zhang, Z. (2012). Memes co-evolution strategies for fast convergence in solving single machine scheduling problems. *International Journal of Production Research, 50*(24), 7357–7377.
Ibaraki, T. (1988). Enumerative approaches to combinatorial optimization - Part iii. *Annals of Operations Research, 11*(1–4), 345–602.
Ibaraki, T., & Nakamura, Y. (1994). A dynamic programming method for single machine scheduling. *European Journal of Operational Research, 76*(1), 72–82.
Ji, M., Chen, K., Ge, J., & Cheng, T. C. E. (2014). Group scheduling and job-dependent due window assignment based on a common flow allowance. *Computers & Industrial Engineering, 68*, 35–41.
Kan, A. R., Lageweg, B., & Lenstra, J. (1975). Minimizing total costs in one-machine scheduling. *Operations Research, 23*(5), 908–927.
Keha, A. B., Khowala, K., & Fowler, J. W. (2009). Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering, 56*(1), 357–367.
Lenstra, J. K., Kan, A. R., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics, 1*, 343–362.
Levner, E., Kats, V., de Pablo, D. A. L., & Cheng, T. C. E. (2010). Complexity of cyclic scheduling problems: A state-of-the-art survey. *Computers & Industrial Engineering, 59*(2), 352–361.
Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). *Iterated local search*. Springer, pp. 254–276.
Ng, C., Wang, J. B., Cheng, T. C. E., & Lam, S. (2011). Flowshop scheduling of deteriorating jobs on dominating machines. *Computers & Industrial Engineering, 61*(3), 647–654.
Pan, Y., & Shi, L. (2007). On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Mathematical Programming, 110*(3), 543–559.
Potts, C. N., & Van Wassenhove, L. N. (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research, 33*(2), 363–377.
Potts, C. N., & Van Wassenhove, L. N. (1991). Single machine tardiness sequencing heuristics. *IIE Transactions, 23*(4), 346–354.
Sen, T., Sulek, J. M., & Dileepan, P. (2003). Static scheduling research to minimize weighted and unweighted tardiness: A state-of-the-art survey. *International Journal of Production Economics, 83*(1), 1–12.
Tanaka, S., Fujikuma, S., & Araki, M. (2009). An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling, 12*(6), 575–593.
Tasgetiren, M. F., Liang, Y. C., Sevkli, M., & Gencyilmaz, G. (2006). Particle swarm optimization and differential evolution for the single machine total weighted

tardiness problem. *International Journal of Production Research, 44*(22), 4737–4754.

Valente, J. M., & Schaller, J. E. (2012). Dispatching heuristics for the single machine weighted quadratic tardiness scheduling problem. *Computers & Operations Research, 39*(9), 2223–2231.

Volgenant, A., & Teerhuis, E. (1999). Improved heuristics for the *n*-job single-machine weighted tardiness problem. *Computers & Operations Research, 26*(1), 35–44.

Wang, X., & Tang, L. (2008). A hybrid VNS with TS for the single machine scheduling problem to minimize the sum of weighted tardiness of jobs. In *Advanced intelligent computing theories and applications. With aspects of artificial intelligence* (pp. 727–733). Springer.

Wang, X., & Tang, L. (2009). A population-based variable neighborhood search for the single machine total weighted tardiness problem. *Computers & Operations Research, 36*(6), 2105–2110.