Applied Soft Computing 62 (2018) 259–271

Contents lists available at ScienceDirect

Applied Soft Computing

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

A hybrid local-search algorithm for robust job-shop scheduling under scenarios

Bing Wang[a,*], Xiaozhi Wang[b], Fengming Lan[a], Quanke Pan[a]

[a] School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200072, China
[b] School of Information Science Technology, East China Normal University, Shanghai 200241, China

ARTICLE INFO

Article history:
Received 31 January 2016
Received in revised form 18 May 2017
Accepted 10 October 2017
Available online 17 October 2017

Keywords:
Robust job-shop scheduling
Scenario
Hybrid local-search algorithm
Simulated annealing
Tabu search

ABSTRACT

This paper discusses an uncertain job-shop scheduling problem with the makespan as the performance criterion. Uncertain processing times are described by discrete scenarios. A robust optimization model is established for the job-shop scheduling problem based on a set of bad scenarios to hedge against the risk of achieving substandard performances among these bad scenarios. To solve the established problem, a problem-specific neighborhood structure is constructed by uniting multiple single-scenario neighborhoods. The constructed neighborhood structure is applied in a hybrid local-search algorithm of combining the simulated-annealing search and the tabu technique. An extensive computational experiment was conducted. The developed algorithm was compared with two possible alternative algorithms. The computational results show the efficiency of the defined neighborhood structure and the competitiveness of the developed hybrid local-search algorithm for the established model.

1. Introduction

Job-shop scheduling problems (JSPs) have been well-known among the hardest combinatorial optimization problems even in a deterministic environment, in which all the data (e.g. the processing times) are assumed to be fixed and precisely known in advance [1]. However, real manufacturing systems could incur various uncertain factors, such as machine breakdowns, worker absenteeism, order changes, etc. These uncertain factors can lead to variations in the processing times. Uncertain JSPs are attracting more and more researchers' attentions.

Uncertain JSPs are more complex than deterministic JSPs. The complexity typically consists in two issues. First, optimization models that are established for uncertain JSPs should reflect certain subjectivity of the decision maker facing uncertain factors. Second, effective and efficient methods that are developed for the established optimization model should especially handle the uncertainty considered.

Uncertain scheduling approaches can be classified by adopted tools of modeling uncertain factors. Stochastic programming is regarded as a typical approach dealing with uncertainty. Stochastic scheduling generally assumes known probability distributions for uncertain factors. Stochastic JSPs have been investigated by many authors [2–5]. However, knowledge about exact probabilistic distributions of uncertain parameters is usually hard to obtain. As an alternative approach to stochastic programming, robust optimization is becoming a promising approach of handling uncertainty [6]. Recently, various robust JSPs are proposed in the literature [7–12].

For robust optimization, scenario approach is one of the important tools to characterize uncertain processing times, in which varying variables in given scenario sets are used instead of random variables with known distributions [13]. For production scheduling problems, the scenario set is usually selected as a continuous interval or a finite set of discrete values [14]. In the interval scenario case, each parameter could take any value between given lower and upper bounds. An interval scenario set actually contains infinite number of scenarios [15–17]. In the discrete scenario case, a number of possible discrete values are listed as a part of the input [18–23]. A discrete scenario set contains finite number of scenarios. When there is no a priori knowledge of which scenario will eventually occur and no probabilities can be assigned to the scenarios, the robust optimization approach consists in searching for "robust" solutions. Intuitively, a solution can be considered as robust if it behaves "not too bad" in all of the scenarios [23].

The attitude toward the risk of performance degradation is an especially important issue in robust scheduling [24]. Considering the risk-averse preference of decision maker, the robust optimization model is established typically by optimizing the scheduling

* Corresponding author.
E-mail address: susanbwang@shu.edu.cn (B. Wang).

https://doi.org/10.1016/j.asoc.2017.10.020
1568-4946/© 2017 Elsevier B.V. All rights reserved.

performance in the worst-case scenario [25]. Most authors used the min-max (regret) criteria to determine robust solutions, which aim to minimize the maximal cost or regret under all of the scenarios. In the interval scenario case, Daniels and Kouvelis [15] proposed the min-max regret model to hedge against the processing time variability in the single-machine scheduling environment. They developed a branch-and-bound algorithm and heuristics to solve the problem. Lu et al. [16] addressed a single-machine scheduling problem with uncertain processing times and sequence-dependent setup times represented by intervals. A simulated annealing-based heuristic was provided as the solution algorithm. Kouvelis et al. [17] discussed the min-max regret model of a two-machine flow shop scheduling with uncertain processing times both in the interval scenario case and in the discrete scenario case. A branch-and-bound algorithm and heuristics were developed. Yang and Yu [21] addressed the same problem as Daniels and Kouvelis [15] but with a discrete, finite set of processing time scenarios rather than interval data. They provided a dynamic programming approach and heuristics. Aissi et al. [26] presented a review on the min-max (regret) robust optimization for combinatorial optimization problems and pointed out that the min-max (regret) robustness has limitations. In some situations, the min-max (regret) robust solutions are too pessimistic for decision makers who are willing to accept some degree of risk. Moreover, the optimal performance on each scenario should be known as a reference standard for regret in the min-max regret model, and this is an intractable issue for NP-hard problems. Some authors tried to look for alternative approaches to obtain less conservative robust solutions. Daniels and Carrillo [18] proposed the beta-scheduling robustness approach through a target performance specified by the decision maker. They developed a branch-and-bound algorithm and heuristics to solve the problem. Kalai et al. [23] proposed the concept of lexicographic a-robustness in the discrete scenario case. More studies on robust optimization approaches can be found in the reviews [27,28].

Much more effort has been made on the robust single-machine scheduling problems or the robust flow-shop scheduling problems. The optimal performance can be obtained easily if the single-machine or flow-shop scheduling problem is polynomially solvable. This could be the reason that the min-max regret model was mainly applied in some single-machine scheduling problems or a few flow-shop scheduling problems. To the best of our knowledge, robust JSPs under scenarios are seldom reported.

It is challenging to develop satisfactory robust optimization approaches for uncertain JSPs. The traditional min-max (regret) robustness has the shortcoming of over-conservatism because great importance is attached to the sole worst-case scenario that is sometimes unlikely to occur. In fact, poor system performances could occur under not only the worst-case scenario but also more other bad scenarios. Moreover, the min-max (regret) model is not applicable for NP-hard JSPs because the optimal performance on individual scenario could not be obtained easily.

The motivation of this paper is to propose a new robust scheduling approach for the JSP with the makespan to hedge against the risk of system performance degradation in the situation that uncertain processing times of jobs are characterized by discrete scenarios. There are four main contributions in this paper. First, we establish a new robust optimization model for the uncertain JSP discussed as an alternative to traditional robust JSPs. In the new robust optimization model, more bad scenarios are concerned than the sole worst-case scenario, and a threshold of performance is given as a reference standard to evaluate system performances instead of the optimal performance. Second, we construct a problem-specific neighborhood structure based on the characteristic of the established robust JSP model. To the best of our knowledge, this research is the first to define a problem-specific neighborhood structure for an uncertain JSP under scenarios. Third, we develop an effective hybrid local-search algorithm to solve the established robust JSP. The developed algorithm outperforms two possible alternative algorithms due to the constructed neighborhood structure and the adopted hybrid algorithm framework. Fourth, from the experimental studies carried out in this paper, we obtained the insights of the established robust JSP model as well as its solution algorithm that may be useful to develop new robust optimization models and solution algorithms.

The remainder of this paper is organized as follows. In Section 2, the literature on existing studies of uncertain JSPs and hybrid algorithms for JSPs is reviewed. In Section 3, the new robust optimization model is established for the scenario JSP with the makespan. In Section 4, a complex neighborhood structure is constructed for the established robust JSP model and exploited in a hybrid local-search algorithm framework. An extensive experiment was conducted to investigate the developed algorithm in Section 5. Last, conclusions and future research directions are given in Section 5.

2. Literature review

In this section, we review the main studies regarding uncertain JSPs and hybrid algorithms developed for JSPs, especially for the most relevant JSP with the makespan.

2.1. Uncertain JSPs

In uncertain environments, stochastic JSPs and robust JSPs are two main types of uncertain JSPs. Various stochastic JSPs and robust JSPs have been dealt with in recent decades in the literature [2–5,7–12].

Stochastic JSPs typically aim to optimize expected system performances. Zhang et al. [2] studied a job shop scheduling problem with random processing times under the objective of minimizing the expected total tardiness. A hybrid differential evolution algorithm was developed for it. Lei et al. [3] proposed a multi-objective job shop scheduling problem with stochastic processing time. The objective is to minimize the makespan and the total tardiness ratio simultaneously. A simplified genetic algorithm was used as solution algorithm. Gu et al. [4] proposed a novel competitive co-evolutionary quantum genetic algorithm for stochastic job shop scheduling problem with the objective to minimize the expected value of makespan. Moghaddam et al. [5] presented a nonlinear mathematical programming model for a stochastic job shop scheduling problem. They proposed a hybrid method based on neural network and simulated annealing (SA), in which a neural network is applied to generate an initial feasible solution and then a SA is used to improve the quality of the initial solution.

Robust JSPs usually aim to optimize certain robustness criteria instead of expected system performances required by stochastic JSPs. Thus robust JSP models are diverse due to various robustness definitions. Adapting to robust JSP models, various solution approaches are developed in the literature [7–10]. Leon et al. [7] proposed a robustness measure for the job-shop scheduling problem with the makespan as the criterion in an uncertain environment with machine breakdowns and processing time variability. They solved the robust job-shop scheduling problem using a genetic algorithm. Byeon et al. [8] handled the robust job-shop scheduling with weighted tardiness as the performance, using decomposition heuristics. Jensen [9] generated robust and flexible job shop schedules using genetic algorithms. Goren et al. [10] addressed stable job-shop scheduling subject to the expected makespan being lower than a threshold T in a similar uncertain environment with processing time variability. They developed a tabu search algorithm with the mean-critical path neighborhood to handle larger instances of the problem.

In addition to stochastic JSPs and robust JSPs, other types of uncertain JSPs are also introduced by recent studies [29,30]. Qiu et al. [29] developed an AIS-based hybrid algorithm for a dynamic job shop scheduling problem. Lei [30] proposed a fuzzy flexible job shop scheduling problem and develop the co-evolutionary genetic algorithm to solve it. However, among these existing studies for uncertain JSPs, we did not find any report on an uncertain JSP under discrete scenarios.

As a preliminary work of this paper, we proposed the concept of bad scenario set in order to hedge against the risk of system performance degradation in the paper [11]. In addition, we used a hybrid genetic simulated-annealing algorithm to solve a two-level robust job-shop scheduling model in the literature [12].

The meta-heuristic approaches are the main methods to handle uncertain JSPs due to the high complexities of JSPs. Among these meta–heuristic approaches developed for uncertain JSPs, hybrid algorithms have played important roles [2,4,5,12,29]. Hybrid algorithms are also extensively investigated for deterministic JSPs. In the following, we will provide a review on hybrid algorithms for deterministic JSPs.

2.2. Hybrid algorithms for JSPs

In addition to those aforementioned hybrid approaches for uncertain JSPs, various hybrid algorithms have been developed for deterministic JSPs [31–47], especially for the JSP with the makespan objective. The JSP with the makespan is well known as one of the hardest problems. Azizi et al. [31], proposed a hybrid method called the adaptive tabu-simulated annealing method for the JSP with the makespan. This hybrid technique takes the advantages of the stochastic simulated annealing to escape local minima and at the same time it improves the performance of the search by having a tabu list on the side. Yusof et al. [32] solved the same job shop scheduling problem using a hybrid parallel micro genetic algorithm. Goncalves et al. [33] developed a hybrid genetic algorithm with tabu search for the JSP. Zhang et al. [34] designed a hybrid algorithm combining Tabu Search (TS) and SA, in which SA is used to construct elite solutions in the promising Big Valley region while TS is subsequently applied for intensification. Ponsich et al. [35] introduced a hybrid algorithm of combining Differential Evolution (DE) and TS for the same JSP. Eswaramurthy et al. [36] hybridized tabu search with ant colony optimization for solving the JSP. Nasiri et al. [37] presents a hybrid algorithm which combines global equilibrium search, path relinking and tabu search to solve the JSP. Wang et al. [38] developed a hybrid genetic simulated annealing algorithm for the deterministic JSP.

Moreover, hybrid algorithms are also developed for other JSPs. Zhang et al. [39] proposed a hybrid simulated annealing algorithm based on a novel immune mechanism for the JSP with the objective of minimizing total weighted tardiness. Mencia et al. [40] designed a memetic algorithm that combines a genetic algorithm with a tabu search algorithm which incorporates a problem-specific neighborhood structure in its core for the JSP with operator. Qing-dao-er-ji et al. [41] proposed an inventory based two-objective job shop scheduling model and designed a hybrid genetic algorithm to solve it. Yuan et al. [42] proposed a hybrid harmony search (HHS) algorithm for solving the flexible job shop scheduling problem with the criterion to minimize the makespan.

Hybrid meta-heuristic algorithms are standing out among various methods for JSPs because they can combine the merits of different meta-heuristic algorithms. Hybrid algorithms often perform remarkably well through incorporating local search to achieve a proper combination of diversification and intensification search efforts. Among the great diversity of studies, the clear superiority of local-search techniques is verified in the literature ([31,33–37,39,40,43–47]). As popular local-search methods, SA and TS have been investigated by extensive researches, which indicate that exploring a problem-specific neighborhood of the new schedule could lead to higher quality solutions. Those recently well-known problem-specific neighborhoods are devised for the deterministic JSP mainly based on the critical block concept [44–47]. Efficient hybrid local-search algorithms with problem-specific neighborhoods have been verified for deterministic JSPs. However, efficient neighborhood structure and hybrid local-search algorithm were seldom ever developed especially for uncertain JSPs.

3. Problem description and formulation

The JSP with the makespan objective is described as follows: n jobs are to be processed on m machines. Each job is processed on each machine exactly one time, which is called an operation. Each machine can process only one job at a time, and preemption is not allowed. The sequence in which a job visits a machine (i.e., the precedence relation of the operations) is known a priori. The processing time of each operation is required. A solution of the JSP is a schedule that consists of a sequence of operations on each machine that satisfies the aforementioned constraints. The optimal solution of the JSP is a schedule that minimizes the makespan, which is the completion time of the last operation. If all of the processing times of the operations are known and fixed, then the JSP is referred to as the deterministic JSP. This deterministic JSP is strongly NP-hard [48].

Here, we study an uncertain JSP that corresponds to the deterministic JSP. All of the processing times of the operations are not fixed and are uncertain due to incomplete/unreliable information or unavoidable stochastic variability. We describe uncertain processing times using discrete scenarios. The discussed problem here is referred to as the scenario job-shop scheduling problem (SJSP).

In the studied SJSP, uncertain processing times are assumed to be independent. We describe them through a set of finite number of scenarios, which is denoted by Λ. We denote the number of all possible scenarios in Λ by $|\Lambda|$. Here, we consider a "pure" discrete scenario situation [14]: there is no a priori knowledge of which scenario would eventually occur and no probabilities can be assigned to the scenarios. Each scenario $\lambda \in \Lambda$ represents a unique set of job processing times. Let $J = \{J_1, J_2, ..., J_n\}$ represent the set of n jobs, and let $M = \left\{M_1, M_2, \ldots, M_m\right\}$ represent the set of m machines. Let S denote the set of all possible schedules of JSP. For any schedule $s \in S$, let $C(s, \lambda)$ represent the makespan of schedule s under scenario λ. In fact, a deterministic JSP can be regarded as a special case of SJSP with certain single scenario.

Instead of the concept of the optimal solution for the deterministic JSP, the robust solution is discussed herein for the SJSP. Motivated by the idea of hedging against the risk of system performance degradation, we plan to establish a new robust scheduling model for the SJSP.

The system performance realized by a schedule varies as the scenario varies and could be very poor under some scenarios. For a schedule, the system performances realized under all possible scenarios can be divided into acceptable performances and unacceptable (substandard) performances. What performances are substandard is determined by a reference standard. Those scenarios under which substandard system performances are realized can be regarded as bad scenarios for the schedule [11]. These bad scenarios form the set of bad scenarios that includes the worst-case scenario.

Here a threshold of performance is given by the decision maker as the reference standard to evaluate system performances. The given threshold is used to identify a set of bad scenarios, under which substandard performances could be achieved. Specifically, let T denote the value of the threshold, and for a schedule $s \in S$, the set of bad scenarios is defined with respect to the value of T as follows:

$$\Lambda_T(s) = \{\lambda | C(s, \lambda) \geq T, \ \ \lambda \in \Lambda\} \qquad (1)$$

The expression (1) indicates that the set of bad scenarios $\Lambda_T(s)$ is a subset of Λ. Those scenarios under which system performances are not better than the value of T are included in $\Lambda_T(s)$. For convenience, we call $\Lambda_T(s)$ the threshold-based bad-scenario set (TBS). In the discrete scenario case, the number of bad scenarios of TBS is restricted to be finite due to a finite number of all possible scenarios of Λ.

The threshold might represent a promised delivery date or a limit on the tolerable waiting time for the discussed problem with the makespan. This value of the threshold could be obtained empirically or in other ways. We do not plan to discuss the methods of obtaining it, i.e., we assume that the value of the threshold can be given a priori by the decision maker. In fact, the given threshold can reflect the subjective preference of the decision maker. According to Daniels et al. [18], the risk-adverse decision maker should give a value of the threshold not better than the optimal expected performance across all possible scenarios.

To hedge against the risk of performance degradation, the decision maker could naturally consider adding an amount of penalties on bad scenarios in the optimization criterion. For a schedule, the degree of performance degradation on each bad scenario can be measured by the deviation between the performance of the schedule under the bad scenario and the value of the threshold. We define the penalty on individual bad scenario by the square of the degree of performance degradation. In the discrete scenario case, each bad scenario of TBS is equally important for accepting an amount of penalty. Here we concern the degree of performance degradation not the realization possibility on individual bad scenario. For schedule $s \in S$, the amount of total penalties on TBS (denoted briefly by PT) is exactly the sum of all the penalties on individual bad scenarios of TBS, which is expressed as follows.

$$PT(s) = \sum\nolimits_{\lambda \in \Lambda_T(s)} [C(s, \lambda) - T]^2 \qquad (2)$$

The squared item in expression (2) aims to intensify the penalties on worse scenarios.

We regard the total penalties on TBS as the objective function to minimize. The expression (2) is taken into account as the robust optimization criterion to formulate the PT-robust scheduling problem (PSP) as follows.

$$(\text{PSP}) \min_{s \in S} PT(s) \qquad (3)$$

The PSP generates PT-robust solutions by minimizing the objective function$PT(s)$. We call the value of $PT(s)$ the PT performance for schedule s. The PT performance is measured based on TBS, and TBS is defined with respect to the value of T. Thus a PSP actually corresponds to a value of T. The value of T is used as both the threshold to identify TBS and the standard to measure the PT performance for a schedule. Based on the concept of bad scenarios, the new robust optimization model PSP is established here. Obviously, the PSP is strongly NP-hard for the SJSP because even the corresponding deterministic JSP (single scenario SJSP) is strongly NP-hard [48].

The PSP is proposed based on the bad scenarios of TBS. This characteristic of the PSP will help to define a problem-specific neighborhood structure for the PSP. To solve the PSP, a hybrid local-search iterative algorithm with the defined neighborhood structure will be developed to adapt to the uncertainty described by scenarios in the following section.

4. Solution algorithm

Inspired by efficient hybrid local-search algorithms for deterministic JSPs, we plan to develop a solution algorithm for the established PSP by hybridizing the SA operator and the TS operator. For the deterministic JSP with the makespan, although SA is not a powerful technique, but the initial solution has little influence on the solution quality [44]. By contrast, TS has powerful local-search ability especially when its neighborhood is constructed based on the critical paths [34,45–47]. However, the choice of an initialization procedure usually has an important influence on the best solution found by a TS algorithm. In fact, hybrid algorithms can exert the complementary strengths of SA and TS for JSPs and overcome the deficiencies of either approach [31,34].

The efficiency of any local-search iterative algorithm depends in principle on its neighborhood structure. In conventional SA algorithms for JSPs, a neighbor solution is usually generated randomly and the size of neighborhood is small. Van Laarhoven et al. [44], applied a SA based approximation algorithm to minimize the makespan for the deterministic JSP. They made a major contribution by introducing the use of the neighborhood structure N1 based on the block concept of critical path. Taillard [45] implemented a tabu search approach that adopts the neighborhood structure defined by Van Laarhoven et al. [44]. The TS algorithm developed by Nowicki and Smutnicki [46] became a breakthrough in the area. The best results have been obtained for the JSP with the makespan by the TS algorithms proposed by Nowicki and Smutnicki [46,47]. Nowicki and Smutnicki [46] defined the N5 neighborhood, which has a smaller size while retaining the efficient exploration property. The N5 neighborhood is also adopted in hybrid algorithms developed for the relevant JSPs by other researchers [35–37,40].

A hybrid local-search iterative algorithm is developed for the PSP in the context of SJSP here. The algorithm core is a tabued simulated annealing local search, which is performed in a specialized neighborhood to adapt to the uncertainty of operation processing times. The adopted neighborhood structure is an extension of the N5 neighborhood, which is constructed in the following.

4.1. The united-scenario neighborhood structure for the PSP

Those efficient neighborhoods that are applied in the deterministic JSP with the makespan are mainly based on the critical block concept. The reason is that only the variation in the sequence of operations on the critical paths can cause the variation in the makespan for the current solution. A potential better neighbor solution can be obtained by changing the sequence of operations on the critical paths. The problem-specific neighborhood structures designed for the deterministic JSP indicate that the objective function of the solved problem should be taken into account when an efficient neighborhood is successfully constructed. Although Goren et al. [10] defined the mean-critical path neighborhood for the discussed uncertain JSP, the mean-critical path neighborhood could not be efficient for the PSP here because their neighborhood definition has nothing to do with the objective function of the PSP. We would consider the objective function of the PSP and construct a problem-specific neighborhood structure for the PSP.

Because the objective function of the PSP is given based on TBS, the variation in the system performance under only those bad scenarios of TBS can cause the variation in the objective value. Therefore, we consider that the neighborhood structure for the PSP should be constructed based on TBS. In the context of SJSP, a single-scenario SJSP is actually a corresponding deterministic JSP. Thus a single-scenario neighborhood can be generated according to an efficient neighborhood definition for the deterministic JSP. Uniting all of the single-scenario neighborhoods, we define a compounded neighborhood structure. We call the newly defined neighborhood

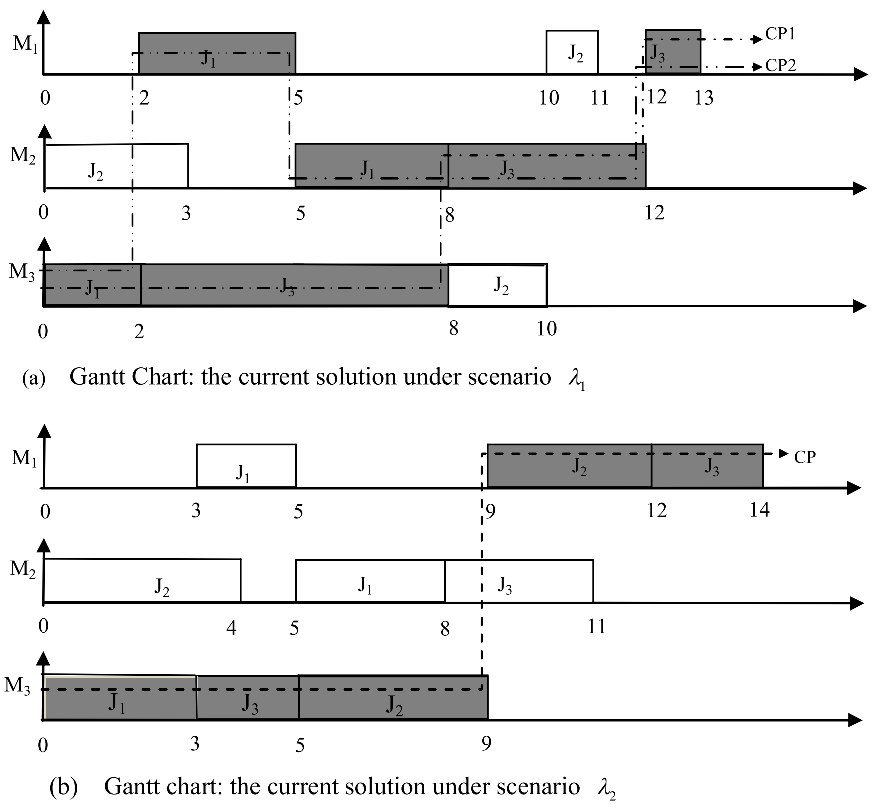

Fig. 1. Gantt charts: the current solution under two scenarios in the 3×3 SJSP example.

the united-scenario neighborhood (UN) because it is constructed by uniting all of the single-scenario neighborhoods that are associated with the bad scenarios of TBS.

Specifically, for the PSP in the context of SJSP, schedule s is evaluated based on TBS with respect to the $PT(s)$ criterion. Because a TBS includes only partial scenarios of Λ, we do not need to consider all possible scenarios of Λ. We denote the single-scenario neighborhood of schedule s under scenario λ by $N(\lambda, s)$. For bad scenario $\lambda \in \Lambda_T(s)$, the single-scenario neighborhood $N(\lambda, s)$ can be generated according to certain neighborhood definition for the deterministic JSP. We denote the union of all of the single-scenario neighborhoods that are associated with the bad scenarios of $\Lambda_T(s)$ by $UN(\Lambda_T(s))$, thus we have

$$UN(\Lambda_T(s)) = \cup_{\lambda \in \Lambda_T(s)} N(\lambda, s) \tag{4}$$

Here we adopt the N5 neighborhood introduced by Nowicki and Smutnicki [46] for the deterministic JSP to generate the single-scenario neighborhood $N(\lambda, s)$. For the PSP in the context of SJSP, UN is an extension of the N5 neighborhood.

Now we recall the related concepts about the N5 neighborhood. The N5 neighborhood is constructed based the concepts of critical path and block. The critical path is the longest path in the realization of the disjunctive graph, i.e. the graph formed by directed edges giving the sequence of operations in each job, plus orientations of the edges between operations on the same machine. A critical path can be decomposed into blocks. A block is a maximal sequence of consecutive operations on a critical path processed on the same machine [46]. Two consecutive blocks contain operations processed on different machines. The N5 neighborhood is generated by swapping the border operation pairs of each block that consist of not less than two operations. Here we refer to an eligible block as a block consisting of at least two operations. In the following, we present the detail procedure for constructing UN.

According to the definition of N5, each candidate solution of $N(\lambda, s)$ is obtained by performing an operator of swapping operation pairs (SOP) for schedule s. In other words, each SOP actually corresponds to a candidate solution of $N(\lambda, s)$. For convenience, we use a SOP to represent a candidate solutions of $N(\lambda, s)$.

Generally, let M_k (p_{ik}) denote the operation O_{ik} of the job J_i on the machine M_k while the processing time of operation O_{ik} is indicated to be p_{ik}. Let $(o_{ik} - o_{jk})$ denote the block that consists of the ordered operations of job J_i and job J_j on the machine M_k, and let (o_{ik}, o_{jk}) denote the SOP that consists of the ordered operations of job J_i and job J_j on the machine M_k. For an example, we give a 3×3 SJSP instance under two scenarios, in which there are the same precedence relations but different operation processing times. Fig. 1 presents the Gantt charts of a current solution for the 3×3 SJSP instance, in which two situations, (a) and (b), are presented to illustrate two scenarios λ_1 and λ_2. In situation (a), we identify two critical paths under scenario λ_1, CP_1 and CP_2, which are marked in Fig. 1(a). In CP_1, there is one eligible block: $(o_{12} - o_{32})$. In CP_2, there are one eligible block: $(o_{13} - o_{33})$. In situation (b), we identify only one critical path under scenario λ_2, CP, which is marked in Fig. 1(b). In CP, there are two eligible blocks: $(o_{13} - o_{33} - o_{23})$ and $(o_{21} - o_{31})$. Three eligible blocks generates five SOPs: (o_{12}, o_{32}),(o_{13}, o_{33}), (o_{13}, o_{33}), (o_{33}, o_{23}) and (o_{21}, o_{31}). Among them, (o_{13}, o_{33}) appears for two times because both $(o_{13} - o_{33})$ and $(o_{13} - o_{33} - o_{23})$ generate the same SOP (o_{13}, o_{33}).

We name repetitive SOP if a SOP appears for more than one time in the union. When we unite all of $N(\lambda, s)$ for $\lambda \in \Lambda_T(s)$ to constitute $UN(\Lambda_T(s))$, repetitive SOPs would generate repetitive candidate solutions, which are evaluated repeatedly and result in unnecessary extra computational burdens. Therefore, repetitive SOP should be removed from the union.

In the example illustrated by Fig. 1, (o_{13}, o_{33}) is one repetitive SOP. Let $Uso(\lambda_1, \lambda_2)$ denote the set of all of the SOPs generated under scenarios λ_1 and λ_2. To remove the repetitive SOPs (o_{13}, o_{33})

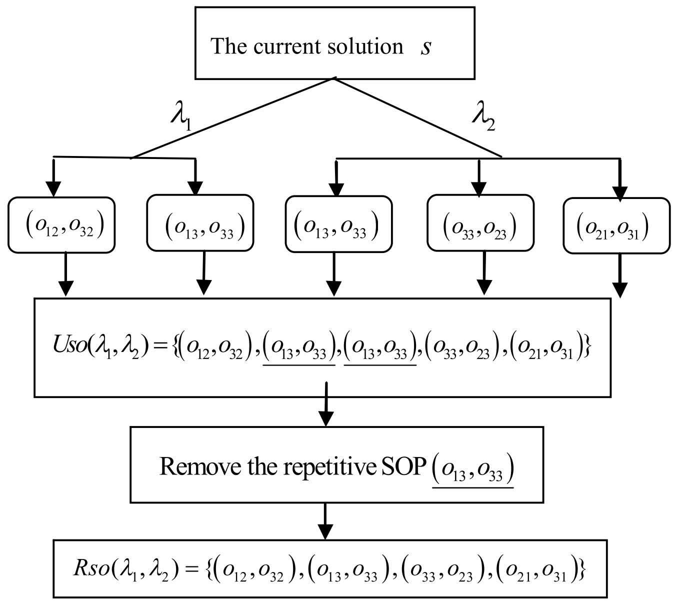
The current solution s
λ_1
λ_2
(o_{12},o_{32})
(o_{13},o_{33})
(o_{13},o_{33})
(o_{33},o_{23})
(o_{21},o_{31})
$Uso(\lambda_1,\lambda_2)=\{(o_{12},o_{32}),(o_{13},o_{33}),(o_{13},o_{33}),(o_{33},o_{23}),(o_{21},o_{31})\}$
Remove the repetitive SOP (o_{13},o_{33})
$Rso(\lambda_1,\lambda_2)=\{(o_{12},o_{32}),(o_{13},o_{33}),(o_{33},o_{23}),(o_{21},o_{31})\}$

Fig. 2. The construction of united-scenario neighborhood in the 3 × 3 SJSP example.

```
Procedure: constructingUN( $\Lambda_T(s)$ )
Input: the current solution $s$ and its bad-scenario set $\Lambda_T(s)$
Output: $UN(\Lambda_T(s))$
Begin
Step 1  For each scenario $\lambda \in \Lambda_T(s)$,
            Generate the block set $B(\lambda,s)$ on different machines.
            Generate the SOP set $SO(\lambda,s)$ from the block set $B(\lambda,s)$ according to N5.
        End for
Step 2  Unite $SO(\lambda,s)$ for all $\lambda \in \Lambda_T(s)$ to generate the SOP union $Uso(\Lambda_T(s))$.
        For $so_1$ is each SOP in $Uso(\Lambda_T(s))$
            For $so_2$ is each other SOP than $so_1$
            If $so_2$ is the same as $so_1$, update $Uso(\Lambda_T(s))$ by removing $so_2$ from
            $Uso(\Lambda_T(s))$;
            End for
        End for
Step 3  Let the initial neighborhood $UN(\Lambda_T(s)) = \varnothing$.
Step 4  For each SOP in $Uso(\Lambda_T(s))$,
            perform the SOP on the current solution $s$ and get a new neighbor solution $so(s)$;
            add the new neighbor solution $so(s)$ into the neighborhood $UN(\Lambda_T(s))$.
        End for
Output  $UN(\Lambda_T(s))$
End
```

Fig. 3. The pseudo-code for the procedure of constructing $UN(\Lambda_T(s))$.

from $Uso(\lambda_1, \lambda_2)$, the reduced union $Rso(\lambda_1, \lambda_2)$ can be obtained. Fig. 2 illustrates the procedure of constructing UN for the example in Fig. 1. Based on $Rso(\lambda_1, \lambda_2)$, $UN(\lambda_1, \lambda_2)$ can be constructed.

Generally, let $Uso(\Lambda_T(s))$ denote the union of SOPs generated based on $\Lambda_T(s)$. $UN(\Lambda_T(s))$ can be obtained by removing repetitive SOPs from $Uso(\Lambda_T(s))$. The procedure for constructing $UN(\Lambda_T(s))$ is realized by the function *constructingUN*$(\Lambda_T(s))$. Fig. 3 presents the pseudo-code for *constructingUN*$(\Lambda_T(s))$ in details.

The united-scenario neighborhood $UN(\Lambda_T(s))$ is constructed especially for the PSP in the context of SJSP. UN is constructed for the current solution based on its bad scenarios, which are directly associated with the objective value of the PSP. Thus UN is a problem-specific neighborhood structure. Comparing with those single-scenario neighborhoods for the deterministic JSP, UN has a larger size. This defined neighborhood structure is applied in a hybrid local-search algorithm framework to solve the PSP of the SJSP.

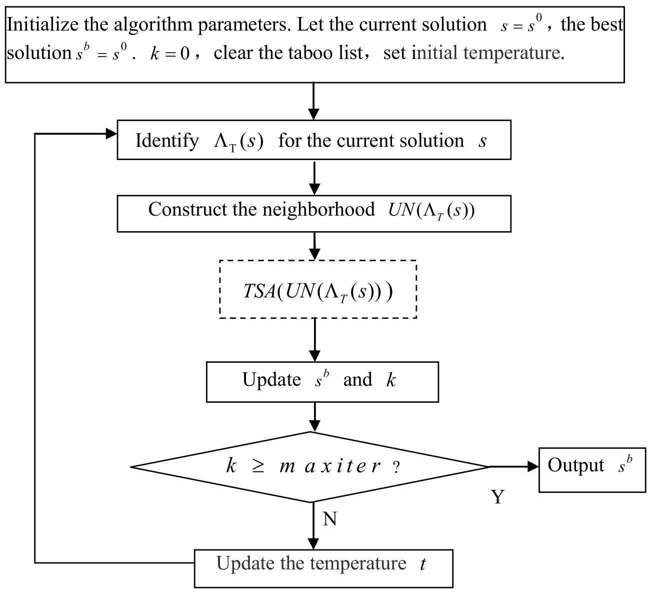
Initialize the algorithm parameters. Let the current solution $s = s^0$, the best solution $s^b = s^0$. $k = 0$, clear the taboo list, set initial temperature.
Identify $\Lambda_T(s)$ for the current solution s
Construct the neighborhood $UN(\Lambda_T(s))$
$TSA(UN(\Lambda_T(s)))$
Update s^b and k
$k \geq maxiter$?
Output s^b
Y
N
Update the temperature t

Fig. 4. The algorithm framework of TSAUN.

4.2. The hybrid local-search algorithm with UN

Due to the large UN structure, we do not need a parallel iterative search but adopt a serial iterative search to build our algorithm framework because the UN structure could allow the algorithm to escape local optima and lead to more efficient exploration in local search for the PSP. However, the cycling and revisiting could probably happen for a large neighborhood structure. Thus we combine the SA operator and the TS operator to perform local search in UN. We call the hybrid algorithm the tabued simulated annealing with UN (TSAUN).

4.2.1. The algorithm framework of TSAUN

The TSAUN algorithm executes a serial iterative procedure and locally search UN for current solution to accept sufficiently“good”solutions. From an initial solution on, at first, $\Lambda_T(s)$ is identified for the current solution s, then $UN(\Lambda_T(s))$ is constructed based on $\Lambda_T(s)$. Local search is performed by a tabued simulated annealing (TSA) search mechanism in $UN(\Lambda_T(s))$. We represent the local-search procedure in $UN(\Lambda_T(s))$ by $TSA(UN(\Lambda_T(s)))$. As $TSA(UN(\Lambda_T(s)))$ is performed, the current solution s is updated iteratively. The algorithm framework of TSAUN is illustrated in Fig. 4, in which $TSA(UN(\Lambda_T(s)))$ is the core part of TSAUN.

4.2.2. The local search of TSAUN

The local search of TSAUN is performed by the procedure $TSA(UN(\Lambda_T(s)))$ at each iteration. The flow chart of $TSA(UN(\Lambda_T(s)))$ is illustrated in details in Fig. 5.

Basically, $TSA(UN(\Lambda_T(s)))$ selects a sufficiently“good”solution from $UN(\Lambda_T(s))$ according to the SA mechanism. SA is a stochastic local-search technique based on principles of physics. The driving mechanism of SA is based on the neighborhood search in which a probability function determines the transition from one solution to another [44]. In the basic form of SA, the algorithm begins to search the solution space by selecting a neighbor from the neighborhood of an initial solution. The Metropolis criterion is used to determine if the selected candidate solution is accepted. The cost value of the candidate solution is compared with that of the current solution. If the candidate solution improves the quality of the current solution, it is accepted and a move is made. Otherwise, a transition probability function would be used to determine whether to accept or to reject the candidate solution. If the value of the transition probabil-

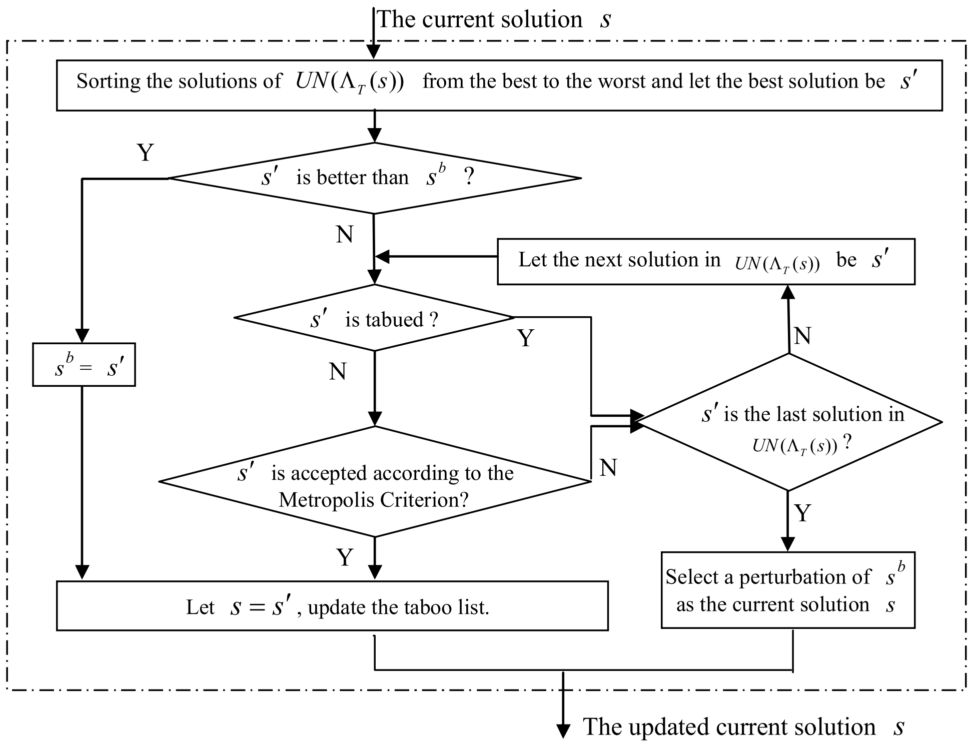
The current solution s
Sorting the solutions of $UN(\Lambda_T(s))$ from the best to the worst and let the best solution be s'
s' is better than s^b ?
Y
N
$s^b = s'$
Let the next solution in $UN(\Lambda_T(s))$ be s'
s' is tabued ?
Y
N
s' is accepted according to the Metropolis Criterion?
N
Y
s' is the last solution in $UN(\Lambda_T(s))$?
N
Y
Select a perturbation of s^b as the current solution s
Let $s = s'$, update the taboo list.
The updated current solution s

Fig. 5. The flow chart of $TSA(UN(\Lambda_T(s)))$.

ity is greater than a random number, then the candidate solution, despite being worse than the current one, is accepted. If the transition to the candidate solution is rejected, another solution from the neighborhood of the current solution would be selected and evaluated.

The search space in a SA algorithm is usually explored one by one and in a sequential manner. Therefore, there is a chance of revisiting a solution multiple times, which leads to extra computation time without improving the quality of the solution. Recently, researchers have attempted to use tabu search with SA to avoid revisiting in deterministic JSP [31]. This technique can also be used in the SJSP.

In TSAUN, UN structure unites the neighbors under multiple scenarios and largely expands the size of neighborhood. For a neighborhood with large size, the exploration ability of local search is obviously intensified. However, at the same time, the possibility of the algorithm's returning to recently visited solutions largely increases. Therefore, it is more necessary to apply the tabu technique to the SA mechanism of local search in order to avoid returning to recently visited solutions.

In the procedure of $TSA(UN(\Lambda_T(s)))$, we first sort the candidate solutions of $UN(\Lambda_T(s))$ from the best to the worst such that the best candidate solution can be selected first. The candidate solution would be checked by the SA operator and the TS operator one by one in this order. A tabu list is appended to the SA operator. The main advantage of adding a tabu list to a SA algorithm is that the search would have a memory that provides an inhibitory feedback to the search. Before an inferior candidate solution of UN is determined by the Metropolis criterion, it would be tested against the tabu list. If such a solution is found in the tabu list, then it is rejected and the next candidate solution from $UN(\Lambda_T(s))$ would be selected. Thus the inferior candidate solution is accepted only in the case that it is passed by both the tabu operator and the SA operator. A situation could arise in which all of the candidate solutions of $UN(\Lambda_T(s))$ are not accepted. In such a situation, our procedure selects a perturbation of the best solution as the current solution.

We define the tabu-list elements to be the attributes of the moves according to the TS procedure [46]. If a move is selected by performing an SOP, then its inverse move is forbidden and added to the tabu list. In general, a longer tabu list is necessary when the size of the neighborhood is larger. In TSAUN, the size of UN is larger than the single-scenario neighborhood. Thus, the tabu-list length should be longer than what is designed for the corresponding deterministic JSP.

In the SA operator, the probability of accepting inferior solutions is proportional to the temperature. The higher the temperature, the greater the probability of receiving inferior solutions is. Traditional SA algorithms usually begin to search at a high temperature. As the temperature gradually decreases, the probability of accepting inferior solutions decreases, and the obtained objective function values become better at the same time. The temperature decreases during the search according to a function known as the cooling schedule or annealing schedule. There are several theoretical and empirical cooling schedules suggested in the literature, among them the monotonic schedule developed by Hajek [49] is the most popular one due to its simplicity and effectiveness. For the deterministic JSP, the adaptive schedules were preferred to by the used SA algorithm [31] because a simple single-scenario neighborhood is adopted. Compared with the single-scenario neighborhood, UN structure enlarges the size of neighborhood and intensifies the local search for the current solution even when the temperature is rather low. Therefore, a simple monotonic cooling schedule could be enough for TSAUN.

The hybrid technique adopted in TSAUN takes the advantages of the stochastic simulated annealing to escape local minima and at the same time it improves the search performance by having a tabu list. This way, the cycling and revisiting is reduced.

4.2.3. The termination criterion of TSAUN

The termination criterion of TSAUN can be given in a similar way as the TS algorithm in [46]. When we perform continuous k ($k \geq maxiter$) steps of moves without improving the best solu-

Table 1
Solutions obtained by TSAUN given different termination criteria in smaller-size instances.

Type	Inst.	T	*maxiter* = 300		*maxiter* = 500		*maxiter* = 1000		*maxiter* = 2000	
			PT	*CPU*	*PT*	*CPU*	*PT*	*CPU*	*PT*	*CPU*
~LA01	1	720	141.5	0.39	131.8	0.68				
	2	728	373.6	0.61	340.6	0.93				
	3	710	135.8	0.42	99.05	0.84				
	4	738	240.5	0.45	190.5	0.56	115.2	1.25		
	5	735	514.2	0.28	471.2	0.56	342.1	1.69		
	6	728	672.8	0.25	589.9	0.49	352.3	0.71	289.6	1.81
	7	701	386.5	0.44	271.3	0.67				
	8	710	316.1	0.31	298.6	0.53	282.4	0.92		
	9	729	195.9	0.49	62.6	0.68				
	10	708	320.5	0.45	298.6	0.70	263.6	1.15		
~LA13	1	1304	961.4	0.79	941.6	0.97	914.6	1.50		
	2	1274	759.8	1.26	623.8	1.62	604.8	1.76	590.0	2.15
	3	1325	837.6	1.19	638.4	2.78				
	4	1266	1196	1.70	1014	2.45	973.1	2.89		
	5	1301	1067	0.85	972.9	1.44	963.2	3.64		
	6	1280	1221	1.28	949.3	1.89				
	7	1279	726.5	0.62	730.5	0.83	715.8	1.23	658.0	1.58
	8	1328	1475	0.61	1328	0.98	1271	1.62		
	9	1308	1397	0.58	1306	1.88				
	10	1310	1227	0.72	1019	1.08	1012	1.95		

tion obtained thus far, the algorithm is terminated with a balance between exploration and intensification steps. Here, *maxiter* is a given number, and the values of *maxiter* could be tuned for specific instances by compromising the solution quality and computational burden in the following section.

In this section, the specialized neighborhood structure UN is constructed for the PSP in the context of SJSP. The TSAUN algorithm is developed as the solution algorithm for the established robust optimization model. To test the effectiveness and the efficiency of the developed algorithm, an extensive experiment was conducted in the following section.

5. Computational results and analysis

We conducted an experiment to investigate the TSAUN algorithm for the PSP. The present section consists of two subsections. In the first subsection, the developed TSAUN was tested under different values of termination criterion for various SJSP instances. In the second subsection, we compared TSAUN with two possible alternative algorithms for the PSP to show the advantages of TSAUN.

In this experiment, test instances were derived from four types of classical job-shop scheduling benchmark problems, including FT10 designed by Fisher and Thompson [50], LA01, LA13, and LA36 designed by Lawrence [51], whose problem sizes are 10*10, 10*5, 20*5 and 15*15 respectively. The processing times of all of the operations were made uncertain using the scenario approach. Each possible processing time was generated identically from the uniform distribution on the interval $[p_{\min}, p_{\max}]$, in which $p_{\min} = 10$ and $p_{\max} = 100$. Let "~FT10", "~LA01", "~LA13", and "~LA36" represent four types of SJSP instances respectively from the benchmark JSPs FT10, LA01, LA13, and LA36. We regard "~LA01" and "~LA13" as the smaller-size instances and "~FT10" and "~LA36" as the larger-size instances in this experiment. Forty instances with $|\Lambda| = 20$ were generated. The initial solutions for the tested algorithms were generated by selecting the best ones out of 20 randomly generated solutions. All of the algorithms were coded in the C language under the Microsoft Visual C++6.0 programming environment. The experiment was conducted on the workstation with Intel Core I7-4790 3.6G CPU with 16.0 G RAM. Due to the nature of heuristic methods, each algorithm that was involved was performed for 20 runs for each instance, and the best solution among the 20 runs was recorded as the result for each algorithm.

5.1. *Testing TSAUN under different values of termination criterion*

First, we tested TSAUN under different values of termination criterion. Appropriate tabu-list lengths were tuned in TSAUN individually for different SJSP types. The parameter of the monotonic cooling schedule is 0.9. Given different values to *maxiter*, we tested the solution qualities obtained by TSAUN for the PSP. A total of 10 instances (Inst.) were generated from each type of SJSP, and a total of 40 instances were generated for four types of SJSP. For each instance, a value of the threshold was given experimentally and we solved the PSP with the given value of the threshold by performing the TSAUN algorithm. The termination criterion was given five different values of *maxiter*. The *PT* performance of solution and the CPU time were recorded for each instance. The computational results for forty instances were reported in Tables 1 and 2.

In Tables 1 and 2, "T" represents the given value of the threshold, "*PT*" represents the *PT* performance expressed by (2), and "*CPU*" represents the CPU time (seconds) that is consumed for obtaining the corresponding solution. Each instance obtained its best solution under certain value of *maxiter*. In Tables 1 and 2, we can see that TSAUN performed quite differently in different instances. In the smaller-size instances "~LA01" and "~LA13", TSAUN obtained their best solutions under the smaller values of *maxiter*. In the larger-size instances, "~FT10" and "~LA36", TSAUN obtained their best solutions under the bigger values of *maxiter*. In these instances, TSAUN could possibly improve their best solutions further if more iteration steps were run. However, we terminated the algorithms under the given values of *maxiter* with the consideration of compromising the solution quality and the CPU time.

The results in Tables 1 and 2 indicate that the appropriate value of the termination criterion depends on problem instance. The computational results that were obtained could provide references to the settings of *maxiter* values for different instances. We adopted these values that were suggested by the results for different types of instances in the following tests.

5.2. *Comparing TSAUN with possible alternative algorithms*

Because the PSP is a new robust scheduling model, there is no existing solution algorithm that could be compared with. We compared TSAUN with two possible alternative algorithms to verify the efficiency of TSAUN. One alternative algorithm is the tabued

Table 2
Solutions obtained by TSAUN given different termination criteria in larger-size instances.

Type	Inst.	T	*maxiter* = 1000		*maxiter* = 2000		*maxiter* = 5000		*maxiter* = 10000	
			PT	*CPU*	*PT*	*CPU*	*PT*	*CPU*	*PT*	*CPU*
~FT10	1	1230	1175	6.34	737.7	20.5	289.4	43.6	123.0	67.8
	2	1232	136.8	7.43	122.5	5.82	114.6	26.1	89.50	48.0
	3	1241	404.6	11.8	384.9	15.6	341.0	25.0	143.6	36.5
	4	1242	70.95	4.79	64.30	15.7	70.95	25.2	57.00	50.5
	5	1263	214.8	8.23	156.8	14.7	127.5	56.9		
	6	1232	188.2	3.98	112.5	5.82	112.5	18.4	92.75	39.0
	7	1219	138.8	2.53	69.25	8.95				
	8	1226	563.2	6.18	415.8	12.9	205.0	27.8	163.8	37.0
	9	1223	193.7	6.48	118.3	11.8	92.75	36.2	78.13	41.1
	10	1243	154.1	9.20	81.75	11.8				

Type	Inst.	T	*maxiter* = 2000		*maxiter* = 5000		*maxiter* = 10000		*maxiter* = 20000	
			PT	*CPU*	*PT*	*CPU*	*PT*	*CPU*	*PT*	*CPU*
~LA36	1	1650	119.1	23.9	88.20	41.1	65.32	69.2	27.81	158
	2	1567	53.60	30.8	45.40	45.9	31.10	159		
	3	1565	97.70	35.4	64.40	75.9	15.60	168		
	4	1532	98.79	37.3	71.56	81.8	29.05	207		
	5	1530	87.15	61.3	58.60	72.2	33.44	98.2	14.15	203
	6	1549	153.6	54.6	81.50	81.1	22.65	196		
	7	1548	118.8	45.8	62.91	68.8	14.25	208		
	8	1579	120.8	52.1	58.25	81.6	25.14	187	19.35	295
	9	1570	297.3	30.7	225.3	75.9	38.05	257		
	10	1540	277.6	62.4	181.7	103	31.25	313		

simulated-annealing algorithm with (traditional) neighborhood (TSAN), which has the same hybrid algorithm framework as TSAUN but has a different neighborhood structure. The neighborhood of TSAN is generated by interchanging the position of two jobs on a randomly selected machine [31]. TSAN is compared with TSAUN to verify the efficiency of the UN structure. Another alternative algorithm is the simulated-annealing algorithm with UN (SAUN), which performs the traditional SA algorithm framework [44] with the same UN structure as TSAUN. SAUN is compared with TSAUN to verify the effectiveness of the tabu technique in TSAUN.

Forty instances that were generated in Subsection 5.1 were tested again in this subsection. Because three algorithms TSAUN, TSAN and SAUN were performed quite differently in different types of instances, the parameters of three algorithms were tuned individually for forty instances. We set the termination criterion of TSAUN according to the results of Tables 1 and 2. The values of termination criterion of TSAN or SAUN were set depending on the instances experimentally. For the other parameters of TSAN, we adopted the same parameter settings as TSAUN. We set the basic parameter settings of SAUN according to Van Laarhoven et al. [44]. For each instance, the PSP with the given value of threshold was solved by TSAUN, TSAN, and SAUN respectively. The computational results are recorded and presented in Figs. 6–9 respectively.

Figs. 6–9 present intuitively the comparisons of three algorithms for different types of instances in terms of two indexes: the *PT* performance and *CPU* times. Figs. 6–9 indicate that the solutions that were obtained by TSAUN are almost uniformly better than those obtained by TSAN or SAUN while TSAUN consumed less CPU time in almost all of the instances. Compared with TSAN, TSAUN obtained much better solutions with obvious less CPU time in almost all of the instances. The computational results demonstrate that TSAUN search locally across the candidate solutions of neighborhood at each iteration extensively and efficiently much more than TSAN. The UN structure that is oriented to TBS can significantly improve the efficiency of the TSA framework. Compared with SAUN, TSAUN also obtained better solutions in almost all of the instances although SAUN also obtained the same good solutions as TSAUN in some instances. Moreover, the CPU time that was consumed by TSAUN was also less than that consumed by SAUN in individual instance.

More specifically, it shows that the degrees of advantage of TSAUN are obviously different for different types of instances. Figs. 6 and 7 present the comparisons of three algorithms for the smaller-size instances "~LA01" and "~LA13", in which the advantages of TSAUN on solution qualities are less obvious than those on CPU times. The reason may be that all of three algorithms can obtain good solutions for the smaller-size instances, as a result, the solution qualities of three algorithms are not obviously different and the advantages of TSAUN are mainly reflected in less CPU times consumed. Figs. 8 and 9 present the comparisons of three algorithms for the larger-size instances "~FT10" and "~LA36", in which the advantages of TSAUN are quite obvious on both solution qualities and CPU times in most instances. It demonstrates that TSAUN performs more efficiently for harder instances.

SAUN outperforms much better than TSAN in most instances. It seems that the UN structure can exert more effect to build an efficient algorithm for the PSP than the tabu technique. For TSAUN and TSAN, the same TSA framework is adopted. Their difference focuses on the neighborhood structure. The specialized UN structure should be the reason that TSAUN outperforms TSAN. To investigate this, both algorithms were tested on a near optimal solution of the first ~FT10 instance. A solution with the makespan 235 was generated and used as the seed solution for both methods. Figs. 10 and 11 respectively illustrate a 3-min search of the two algorithms starting from this point. TSAN did not improve the seed solution again, whereas TSAUN improved the seed solution again and again until a better solution with the makespan 123 was obtained.

The computational results indicate that TSAUN is significantly superior to two alternative algorithms. To obtain the insights into the convergence behaviors of three algorithms, further examination of the convergence curves reveals their differences. Fig. 12 illustrates the convergence curves of three algorithms in the first instance of FT10.

In summary, among three algorithms, TSAUN or SAUN is obviously better than TSAN for the PSP because the adopted UN structure make the local search of the hybrid algorithm largely intensified. Compared with SAUN, TSAUN performs significantly better especially in the larger-size instances "~FT10" and "~LA36" than in the smaller-size instances "~LA01" and "~LA13". The rea-

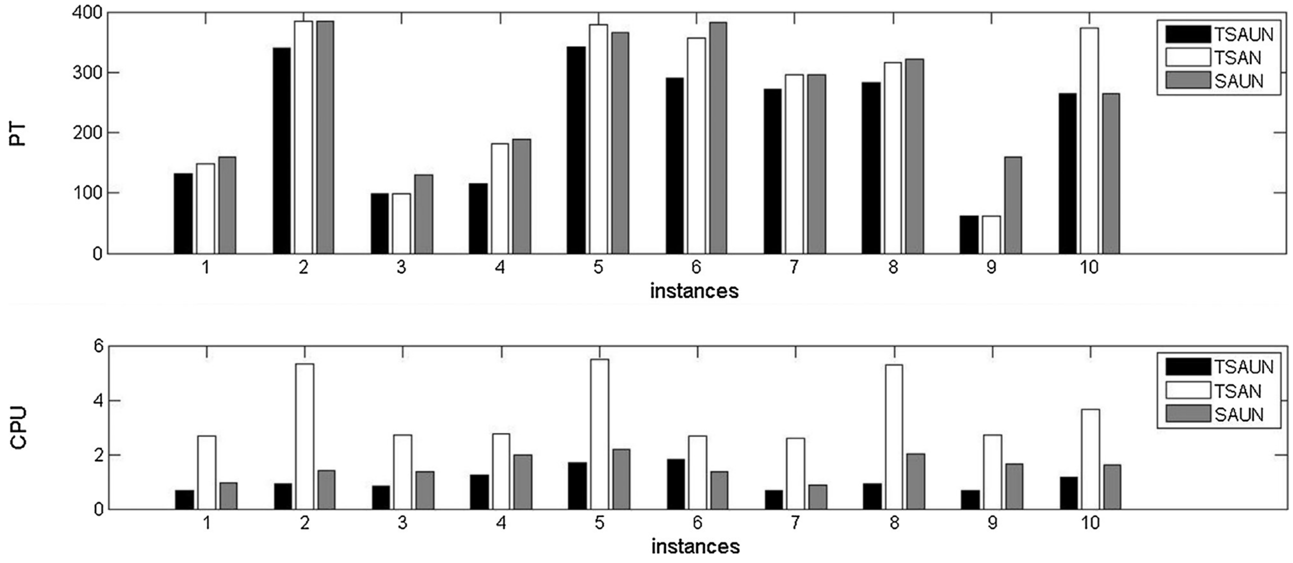

Fig. 6. Comparisons of three algorithms for the PSP in ten ~LA01 instances.

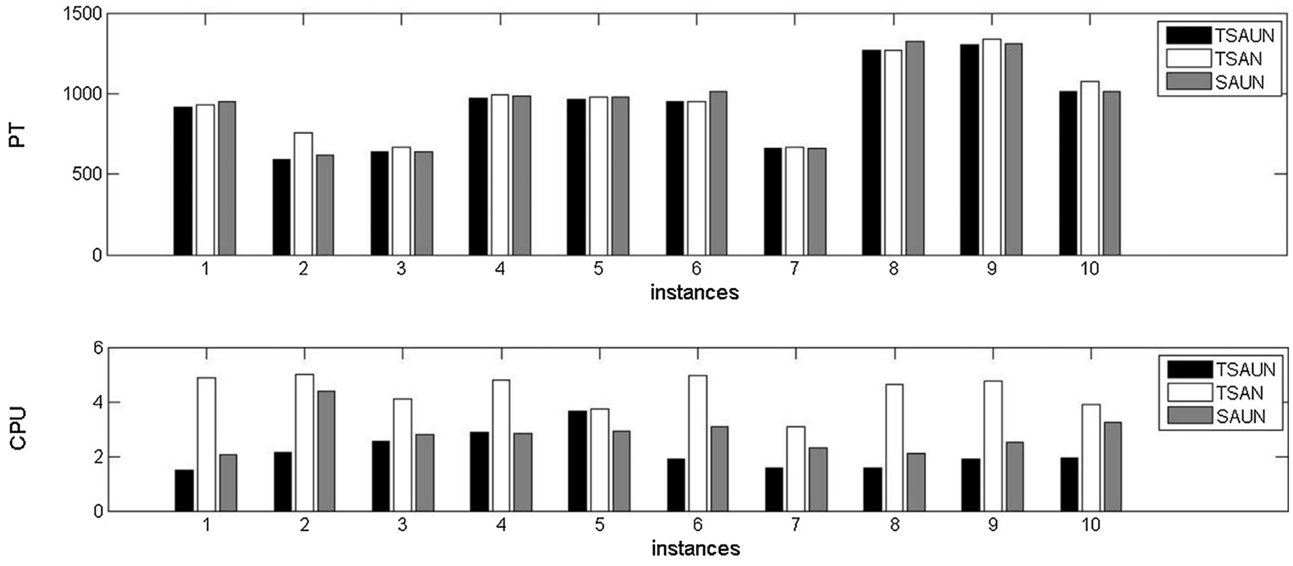

Fig. 7. Comparisons of three algorithms for the PSP in ten ~LA13 instances.

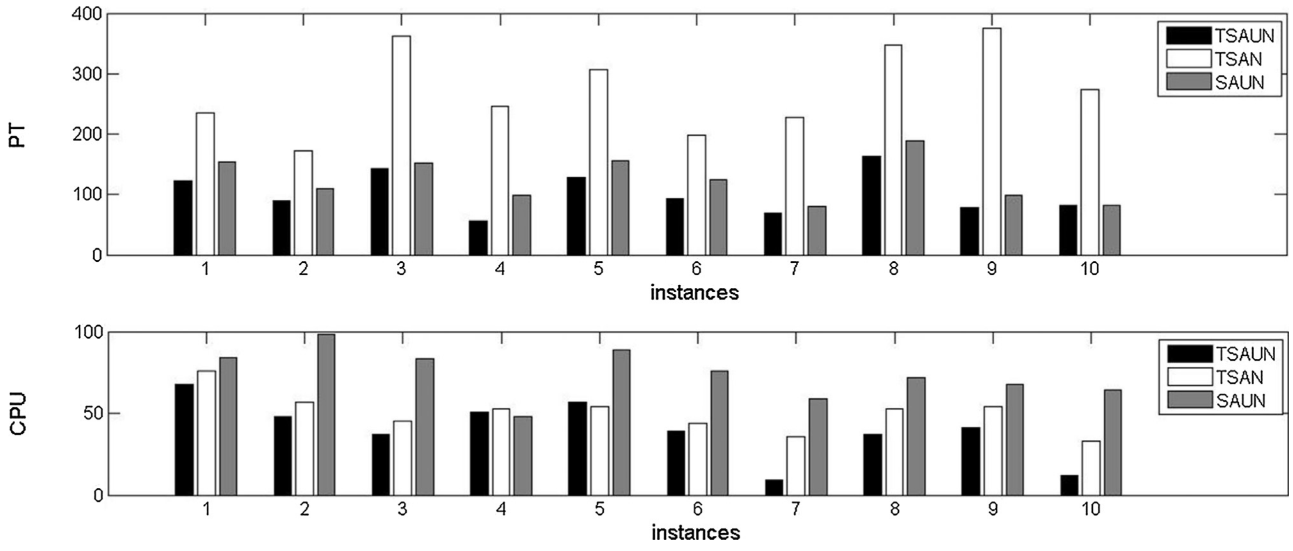

Fig. 8. Comparisons of three algorithms for the PSP in ten ~FT10 instances.

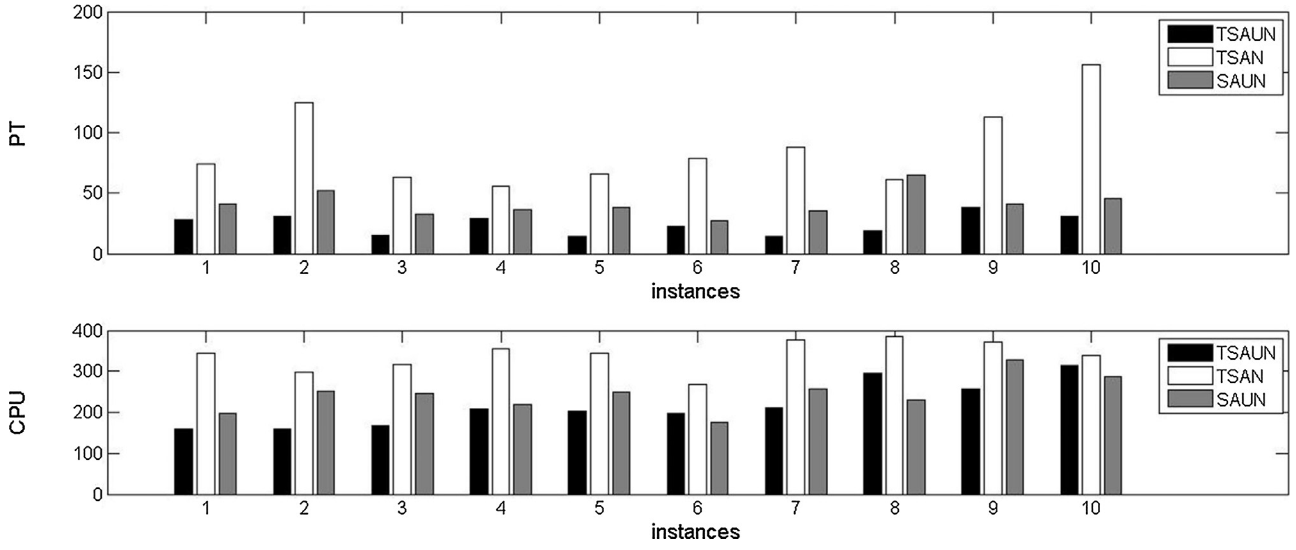

Fig. 9. Comparisons of three algorithms for the PSP in ten ~LA36 instances.

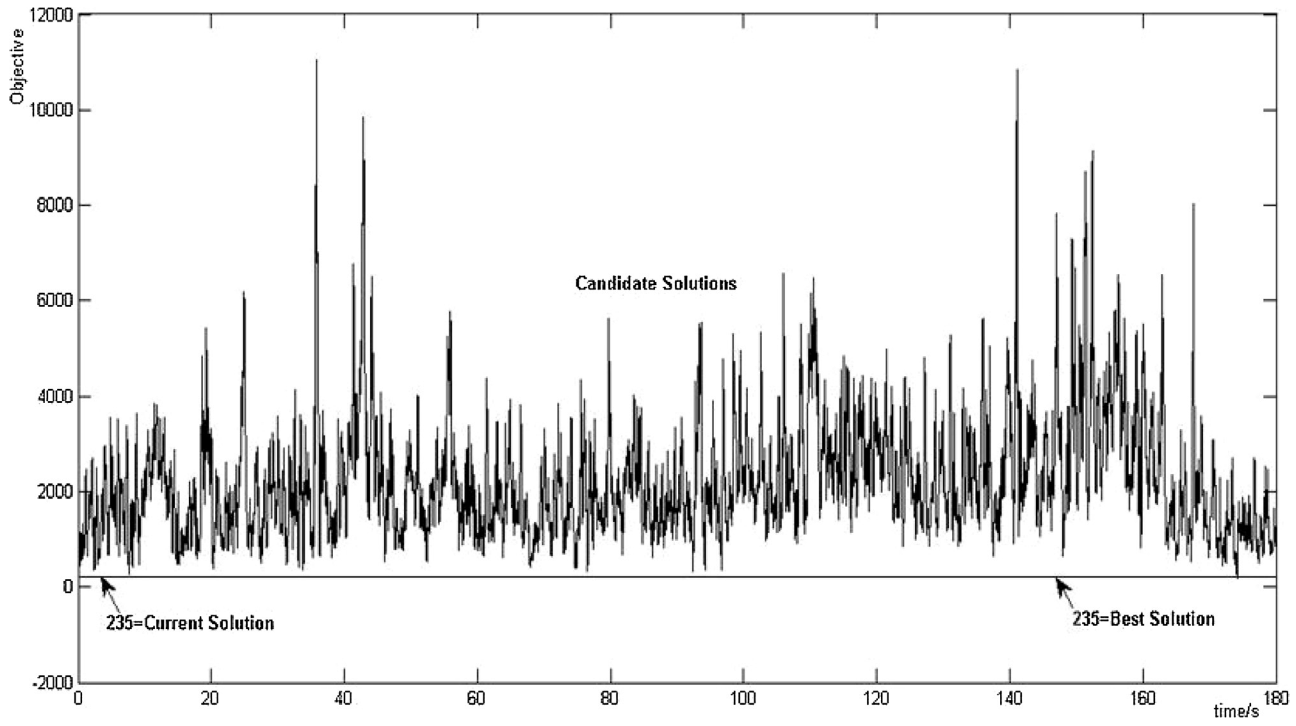

Fig. 10. The chart of TSAN in the first ~FT10 instance.

son is that the tabu technique helps to improve the SA algorithm framework with UN especially for the larger-size instances. The computational results in this subsection illustrate that TSAUN is a better choice for the PSP than two alternative algorithms.

6. Conclusions and future research

This paper discusses the job-shop scheduling problem with the makespan as the system performance, in which uncertain processing times are described by discrete scenarios. The PSP proposed here is a new robust optimization model that is established based on the concept of a bad-scenario set. Given a performance threshold, a threshold-based bad-scenario set is defined.

Based on the characteristics of the PSP, the UN structure is constructed by uniting all of the single-scenario neighborhoods that are associated with the bad scenarios of TBS. Based on UN, the hybrid algorithm TSAUN is developed to solve the PSP. The framework of TSAUN performs a core of SA and applies the tabu technique in local search. This hybrid algorithm takes the advantages of the stochastic simulated annealing to escape local minima and at the same time it improves the search performance by having a tabu list. Extensive experimentation was conducted. TSAUN was compared with two possible alternative algorithms. The computational results demonstrate that TSAUN is effective for the PSP, and it obviously outperformed two alternative algorithms in almost all of the test instances. The computational results verified the efficiency of the defined UN structure as well as the developed hybrid algorithm TSAUN. In summary, we provide a new and effective robust scheduling approach for the SJSP in this paper.

Future research could be done in the following directions:

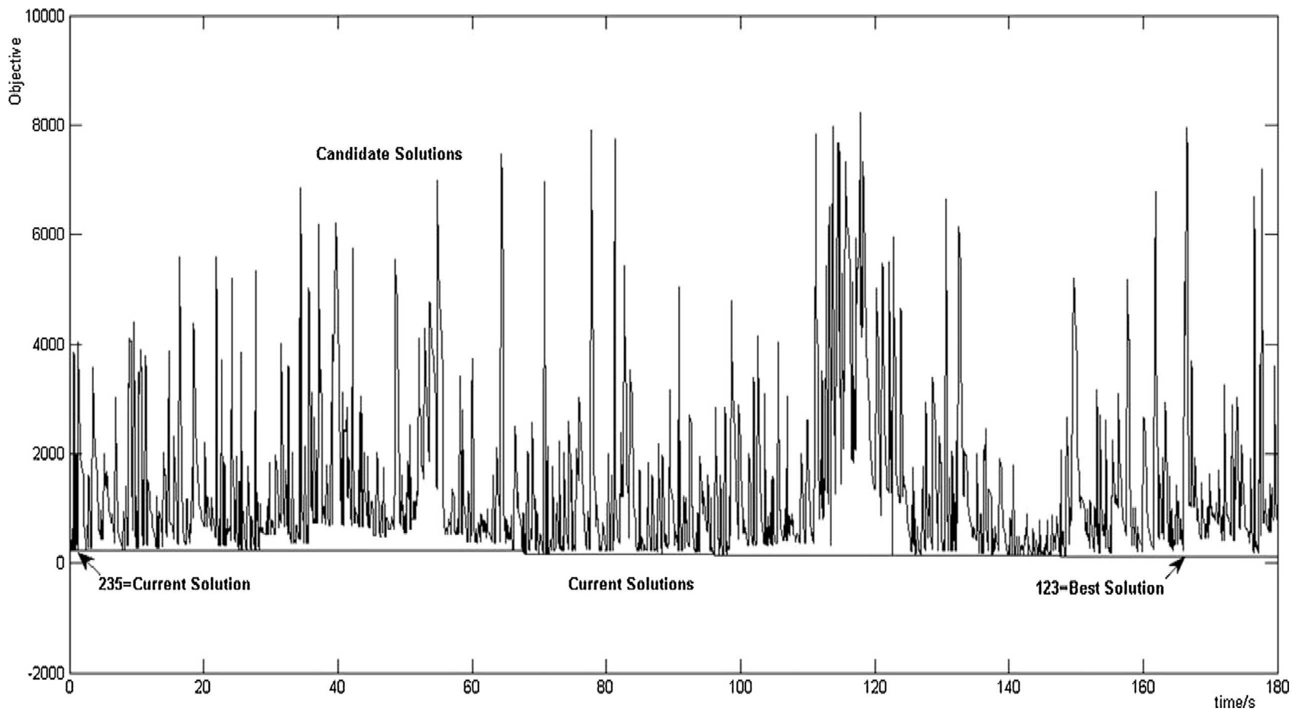

Fig. 11. The chart of TSAUN in the first ~FT10 instance.

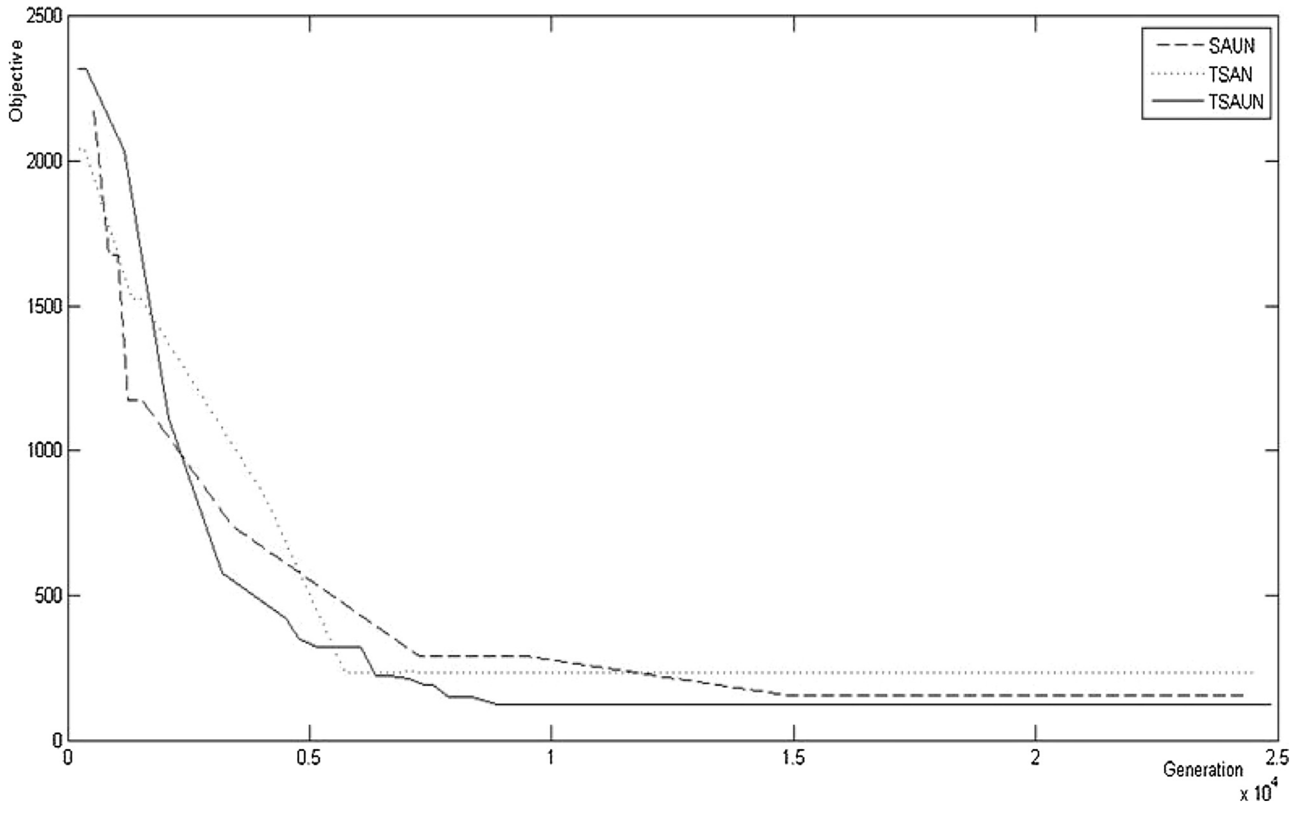

Fig. 12. Comparisons of convergence curves of three algorithms in the first ~FT10 instance.

(1) In this paper, the *PT*-robust solutions are obtained given a value of threshold in advance. This is a single-stage decision framework. If the value of the threshold is not given a priori by the decision maker, the situation in which the value of the threshold varies could be discussed in a two-stage decision framework. This could be a future research direction. *PT*-robust solutions can be obtained for different values of threshold.

(2) The single-scenario neighborhood is generated only with respect to the N5 structure in this paper. More neighborhood definitions for single scenario could be adopted

to construct various complex neighborhood structures for multiple-scenario uncertain environments in the future.

(3) The PSP model could be applied to more discrete optimization problems with the uncertainty characterized by discrete scenarios, and more efficient algorithms could be developed to address the PSP in various problem contexts.

Acknowledgment

This work is partly supported by National Natural Science Foundation of China (#60874076, #61273040 and #61773252).

References

[1] A.S. Jain, S. Meeran, Deterministic job-shop scheduling: past, present and future, Eur. J. Oper. Res. 113 (2) (1999) 390–434.
[2] R. Zhang, S.J. Song, C. Wu, A hybrid differential evolution algorithm for job shop scheduling problems with expected total tardiness criterion, Appl. Soft Comput. 13 (2013) 1448–1458.
[3] D. Lei, Simplified multi-objective genetic algorithms for stochastic job shop scheduling, Appl. Soft Comput. 11 (2011) 4991–4996.
[4] J.W. Gu, M. Gu, X.S. Gu, A novel competitive co-evolutionary quantum genetic algorithm for stochastic job shop scheduling problem, Comput. Oper. Res. 37 (2010) 927–937.
[5] R.T. Moghaddam, F. Jolai, F. Vaziri, P.K. Ahmed, A. Azaron, A hybrid method for solving stochastic job shop scheduling problem, Appl. Math. Comput. 170 (2005) 185–206.
[6] J.M. Murvey, R.J. Vanderbei, S.A. Zenios, Robust optimization of large-scale systems, Oper. Res. 43 (2) (1995) 264–281.
[7] V.J. Leon, S.D. Wu, R.H. Storer, Robustness measures and robust scheduling for job shops, IIE Trans. 26 (5) (1994) 32–43.
[8] E.S. Byeon, S.D. Wu, R.H. Storer, Decomposition heuristics for robust job-shop scheduling, IEEE Trans. Rob. Autom. 14 (2) (1998) 303–313.
[9] M.T. Jensen, Improving robustness and flexibility of tardiness and total flow time job shops using robustness measures, Appl. Soft Comput. 1 (2001) 35–52.
[10] S. Goren, I. Sabuncuoglu, U. Koc, Optimization of schedule stability and efficiency under processing time variability and random machine breakdowns in a job shop environment, Nav. Res. Logist. 59 (1) (2012) 26–38.
[11] B. Wang, X.F. Yang, Q.Y. Li, Bad-scenario based robust scheduling model, Acta Autom. Sin. 38 (2) (2012) 270–278.
[12] B. Wang, X.F. Yang, Q.Y. Li, Genetic simulated-annealing algorithm for robust job shop scheduling. Fuzzy Information and Engineering, Vol. 2, Adv. Intell. Soft Comput. 62 (2009) 817–827.
[13] P.J.H. Schoemaker, Multiple scenario development: its conceptual and behavioral foundation, Strat. Manage. J. 14 (3) (1993) 193–213.
[14] P. Kouvelis, D. Yu, Robust Discrete Optimization and Its Application, Kluwer Academic Publisher, 1997.
[15] R.L. Daniels, P. Kouvelis, Robust scheduling to hedge against processing time uncertainty in single-stage production, Manage. Sci. 41 (2) (1995) 363–376.
[16] C.C. Lu, S.W. Lin, K.C. Ying, Minimizing worst-case regret of makespan on a single machine with uncertain processing and setup times, Appl. Soft Comput. 23 (2014) 144–151.
[17] P. Kouvelis, R.L. Daniels, G. Vairaktarakis, Robust scheduling of a two-machine flow shop with uncertain processing times, IIE Trans. 32 (2000) 421–432.
[18] R.L. Daniels, J.E. Carrillo, Beta-scheduling for single-machine systems with uncertain processing times, IIE Trans. 29 (11) (1997) 977–985.
[19] T. Assavapokee, M.J. Realff, J.C. Ammons, I.H. Hong, Scenario relaxation algorithm for finite scenario-based min–max regret and min–max relative regret robust optimization, Comput. Oper. Res. 35 (6) (2008) 2093–2102.
[20] D.S. Yamashita, V.A. Armentano, M. Laguna, Robust optimization models for project scheduling with resource availability cost, J. Scheduling 10 (2007) 67–76.
[21] J. Yang, G. Yu, On the robust single machine scheduling problem, J. Comb. Optim. 6 (2002) 17–33.
[22] A. Kasperski, A. Kurpisz, P. Zielinski, Approximating a two-machine flow shop scheduling under discrete scenario uncertainty, Eur. J. Oper. Res. 217 (2012) 36–43.
[23] R. Kalai, C. Lamboray, D. Vanderpooten, Lexicographic a-robustness: an alternative to min–max criteria, Eur. J. Oper. Res. 220 (2012) 722–728.
[24] Z. Li, M.G. Ierapetritou, Robust optimization for process scheduling under uncertainty, Ind. Eng. Chem. Res. 47 (12) (2008) 4148–4157.
[25] A. Ben-Tal, L.E. Ghaoui, A. Nemirovski, Robust Optimization, Princeton University Press, 2009.
[26] H. Aissi, C. Bazgan, D. Vanderpooten, Min-max and min-max regret versions of combinatorial optimization problems: a survey, Eur. J. Oper. Res. 197 (2009) 427–438.
[27] I. Sabuncuoglu, S. Goren, Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research, Int. J. Comput. Integr. Manuf. 22 (2) (2009) 138–157.
[28] V. Gabrel, C. Murat, A. Thiele, Recent advances in robust optimization: an overview, Eur. J. Oper. Res. 235 (2014) 471–483.
[29] X. Qiu, H.Y. Lau, An AIS-based hybrid algorithm with PDRs for multi-objective dynamic online job shop scheduling problem, Appl. Soft Comput. 13 (2013) 1340–1351.
[30] D. Lei, Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling, Appl. Soft Comput. 12 (2012) 2237–2245.
[31] N. Azizi, S. Zolfaghari, Adaptive temperature control for simulated annealing: a comparative study, Comput. Oper. Res. 31 (2004) 2439–2451.
[32] R. Yusof, M. Khalid, G.T. Hui, S.M. Yusof, M.F. Othman, Solving job shop scheduling problem using a hybrid parallel micro genetic algorithm, Appl. Soft Comput. 11 (8) (2011) 5782–5792.
[33] J.F. Goncalves, J.J. de Magalhaes Mendes, M.G.C. Resende, A hybrid genetic algorithm for the job shop scheduling problem, Eur. J. Oper. Res. 167 (1) (2005) 77–95.
[34] C.Y. Zhang, P.G. Li, Y.Q. Rao, Z.L. Guan, A very fast TS/SA algorithm for the job shop scheduling problem, Comput. Oper. Res. 35 (2008) 282–294.
[35] A. Ponsich, C.A.C. Coello, A hybrid differential evolution-tabu search algorithm for the solution of job-shop scheduling problems, Appl. Soft Comput. 13 (2013) 462–474.
[36] V.P. Eswaramurthy, A. Tamilarasi, Hybridizing tabu search with ant colony optimization for solving job shop scheduling problems, Int. J. Adv. Manuf. Technol. 40 (2009) 1004–1015.
[37] M.M. Nasiri, F. Kianfar, A GES/TS algorithm for the job shop scheduling, Comput. Ind. Eng. 62 (2012) 946–952.
[38] L. Wang, D. Zheng, An effective hybrid optimization strategy for job-shop scheduling problems, Comput. Oper. Res. 28 (2001) 585–596.
[39] R. Zhang, C.A. Wu, Hybrid immune simulated annealing algorithm for the job shop scheduling problem, Appl. Soft Comput. 10 (2010) 79–89.
[40] R. Mencia, M.R. Sierra, C. Mencia, R. Varela, Memetic algorithms for the job shop scheduling problem with operators, Appl. Soft Comput. 34 (2015) 94–105.
[41] Y.P. R.Qing-dao-er-ji, X.L. Wang, Wang Inventory based two-objective job shop scheduling model and its hybrid genetic algorithm, Appl. Soft Comput. 13 (2013) 1400–1406.
[42] Y. Yuan, H. Xu, J.D. Yang, A hybrid harmony search algorithm for the flexible job shop scheduling problem, Appl. Soft Comput. 13 (2013) 3259–3272.
[43] A.P. Rifai, H.T. Nguyen, S.Z.M. Dawal, Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling, Appl. Soft Comput. 40 (2016) 42–57.
[44] P.J.M. Van Laarhoven, E.H.L. Aarts, J.K. Lenstra, Job shop scheduling by simulated annealing, Oper. Res. 40 (1) (1992) 113–125.
[45] E.D. Taillard, Parallel Taboo Search techniques for the job shop scheduling problem, ORSA J. Comput. 6 (2) (1994) 108–117.
[46] E. Nowicki, C. Smutnicki, A fast taboo search algorithm for the job shop problem, Manage. Sci. 42 (6) (1996) 797–813.
[47] E. Nowicki, C. Smutnicki, An advanced Tabu Search algorithm for the job shop problem, J. Scheduling 8 (2) (2005) 145–159.
[48] J.K. Lenstra, A.H.G. Rinnooy Kan, Computational complexity of discrete optimization problems, Ann. Discrete Math. 4 (1979) 121–140.
[49] B. Hajek, Cooling schedules for optimal annealing, Math. OR 13 (1988) 311–329.
[50] J.F. Muth, G.L. Thompson, Industrial Scheduling, Prentice-Hall, Englewood Cliffs, NJ, 1963.
[51] S. Lawrence, Resource Constrained Project Scheduling: an Experimental Investigation of Heuristic Scheduling Techniques (Supplement). Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburg, PA, 1984.