Computers & Operations Research 40 (2013) 2817–2825

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

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

Effect of solution representations on Tabu search in scheduling applications

Chen-Fu Chen, Muh-Cherng Wu*, Keng-Han Lin

Department of Industrial Engineering and Management, National Chiao Tung University, 1001 University Road, Hsinchu 300, Taiwan

ARTICLE INFO

Available online 20 June 2013

Keywords:
Tabu search
Scheduling
Solution representation
Flow shop

ABSTRACT

This research investigates the application of meta-heuristic algorithms to a scheduling problem called permutation manufacturing-cell flow shop (PMFS) from two perspectives. First, we examine the effect of using different solution representations (S_{new} and S_{old}) while applying Tabu-search algorithm. Experimental results reveal that *Tabu*_S_{new} outperforms *Tabu*_S_{old}. The rationale why *Tabu*_S_{new} is superior is further examined by characterizing the intermediate outcomes of the evolutionary processes in these two algorithms. We find that the superiority of S_{new} is due to its relatively higher degree of freedom in modeling *Tabu* neighborhood. Second, we propose a new algorithm *GA_Tabu*_S_{new}, which empirically outperforms the state-of-the-art meta-heuristic algorithms in solving the PMFS problem. This research highlights the importance of solution representation in the application of meta-heuristic algorithm, and establishes a significant milestone in solving the PMFS problem.

1. Introduction

Meta-heuristic algorithms have been widely used in solving complex space-search problems. Such algorithms are essentially based on an evolutionary paradigm [1–11]. That is, a set of solutions are iteratively updated by an evolutionary mechanism until a satisfactory solution is obtained. Examples of meta-heuristic algorithms include genetic algorithm (*GA*), Tabu-search, ant colony optimization (*ACO*), simulated annealing, etc. [12–19].

In a recent study on scheduling, we found that the solution representation scheme of a meta-heuristic algorithm may have a significant effect on the scheduling performance [20]. The scheduling problem is called permutation manufacturing-cell flow shop (PMFS). Two different solution representations (S_{old} and S_{new}) while applying *GA* and *ACO* to solve the PMFS problem are compared. Experimental results indicate that *GA*_S_{new} outperforms *GA*_S_{old}, and *ACO*_S_{new} outperforms *ACO*_S_{old}. Notice that *GA*_S_{old} developed by Lin et al. [3] was the state-of-the-art benchmark.

As extension of the aforementioned study [20], this paper has two research objectives. The first objective is to study the effect of using S_{old} and S_{new} while applying the Tabu-search algorithm to solve the PMFS problem (i.e., comparing *Tabu*_S_{new} against *Tabu*_S_{old}). The second objective is the development of a meta-heuristic algorithm that outperforms all the other meta-heuristic algorithms to date in solving the PMFS problem.

The first research objective leads to the following findings. Experimental results indicate that *Tabu*_S_{new} outperforms *Tabu*_S_{old}. The reason why *Tabu*_S_{old} appears less effective is due to that it has a higher probability of being trapped into a loop while searching solutions. In addition, such a higher tendency to be trapped into a loop is due to that the space spanned by *Tabu*_S_{old} has a relatively lower degree of freedom.

The second research objective leads to the development of two new meta-heuristic algorithms *GA_Tabu*_S_{new} and *GA_Tabu*_S_{old} for solving the PMFS problem. Experimental results indicate that *GA_Tabu*_S_{new} outperforms *GA_Tabu*_S_{old}. This finding gives further empirical supports to the superiority of S_{new} over S_{old}. In addition, *GA_Tabu*_S_{new} outperforms all the other meta-heuristic algorithms, including the two state-of-the-art algorithms *GA*_S_{old} [3] and *GA*_S_{new} [20].

The remainder of this paper is organized as follows. Section 2 describes the PMFS scheduling problem and relevant literature. Section 3 reviews various chromosome representations in solving scheduling problems. Section 4 presents the two solution representation schemes (S_{old} and S_{new}). Section 5 describes the commonality and distinction of the two Tabu-search algorithms (*Tabu*_S_{old} and *Tabu*_S_{new}); their experimental results are in Section 6 and the reasons why *Tabu*_S_{new} outperforms *Tabu*_S_{old} are described in Section 7. In Section 8, we present *GA_Tabu*_S_{new} and *GA_Tabu*_S_{old} and the experimental results for supporting their merits. Conclusions are given in Section 9.

2. Scheduling problem and prior research

The PMFS scheduling problem has been examined by a few studies [3,20,21–28]. The objective function is to minimize makespan.

* Corresponding author. Tel.: +886 3 5731913; fax: +886 3 5729101.
E-mail addresses: tom.iem96g@nctu.edu.tw (C.-F. Chen), mcwu@mail.nctu.edu.tw (M.-C. Wu), qa123654@hotmail.com (K.-H. Lin).

0305-0548/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2013.06.003

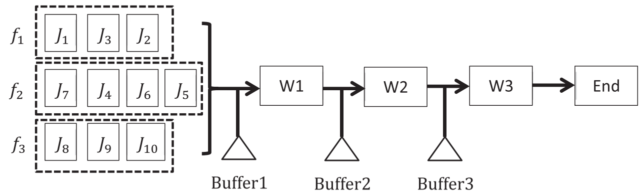

Fig. 1. A permutation manufacturing-cell flow shop.

This section first presents the major assumptions of the scheduling problem, and then proceeds to address the sequencing decisions of the problem. Finally, a real example of the PMFS scheduling problem is presented.

First, it is a *multi-stage flow shop* in which each stage involves only one machine and one buffer with unlimited storage size for storing waiting jobs (Fig. 1). Each machine is so reliable and involves no breakdown in the scheduling horizon.

Second, the shop adopts a *family-based scheduling paradigm*. All jobs are grouped into various families a priori. Each job family is processed in a group manner. Once a job family starts processing in a stage, the stage cannot switch to process any other family unless all jobs in the present family have completed their operations. The shop is a *permutation* flow shop—the job sequence within each family and the sequence among families keep the same for each stage of the shop.

Third, jobs within a family are processed sequentially without requiring new setups of the machines. In contrast, the setup times among families are substantial and *sequence-dependent*. The setup time required for switching to process a new family (say, F_1) depends upon how much is the degree of similarity between family F_1 and its preceding family (say, F_2). For example, the setup time required for undergoing a family switch $F_2 \rightarrow F_1$ may be different from that of $F_3 \rightarrow F_1$.

Fourth, each job is *individually transported*. The jobs of a family are not transported in a batch. Once a job completes its operation at a stage, it is *immediately* and *individually* transported to the buffer of the next stage.

The PMFS scheduling problem has two sequencing decisions: *among-family* sequencing and *within-family* sequencing. Within-family sequencing deals with the sequence of jobs within each family; and among-family sequencing addresses the sequence of families. The within-family sequencing decision is required only when the aforementioned *individual-transportation feature* is strictly imposed in the scheduling problem. Such a sequencing decision would not be needed if the jobs are transported in a batch, because in that case changing job sequence within a family would not change the ultimate scheduling performance.

A real example of the PMFS scheduling problem can be referred to the surface mounting machine (SMT) process that is for mounting electronic components on printed circuit board [21]. In practice, a SMT process is a flow shop, involving a sequence of machines designed for the surface mounting tasks. Each SMT machine is a workstation (or stage) responsible for mounting a particular group of electronic parts on the printed circuit (PC) board, and the group of parts could be changed by a setup. A PC board is taken as a job, which should go through the flow shop to complete all its part mountings. Two jobs involving the same or highly similar part profiles could consecutively go through the flow shop without requiring any setup. Therefore, PC boards are grouped into families and the family-based scheduling paradigm is usually adopted. That is, for each workstation, a significant setup time is urged if it switches to process a new family. The less similar the two consecutive families, the longer the setup time required. This implies that the family setup time is *sequence-dependent*.

The PMFS scheduling problem has been solved by various meta-heuristic algorithms such as Tabu-search algorithm [23], genetic algorithm (*GA*) [25], memetic algorithm [25], and simulated annealing algorithm [26]. Of these meta-heuristic algorithms, the *GA* (called GA_S_{old}) is the state-of-the-art algorithm in 2009. And we had proposed a GA_S_{new} algorithm [20] which outperforms GA_S_{old}.

3. Review on chromosome representations

This section reviews literature that uses different chromosome representations in a particular meta-heuristic algorithm. A pioneer work on different representations by Rothlauf and Goldberg [2] is firstly introduced. Then, example studies [29–31] that use different representations in the application of meta-heuristic algorithms to solve scheduling problems are discussed.

Rothlauf and Goldberg [2] comprehensively investigated the effect of chromosome representation on the performance of meta-heuristic algorithms (also called evolutionary algorithms). In their work, a *chromosome* (i.e., a solution) is represented by a sequence of *genes*, and a gene is represented by a sequence of *alleles*. As shown in Fig. 2, the chromosome involves two genes and each gene is represented by two alleles. The value of each allele is either 0 or 1. The value that represents a gene is shown Table 1, which indicates that each gene value has two gene representations. As a result, a solution has multiple chromosome representations. This property is called redundancy of encoding. In literature, a chromosome is called genotype, while a solution is called phenotype [2,32]. According to this definition, the chromosomes in Fig. 2 involve 16 genotypes and four phenotypes.

In applying meta-heuristic algorithms to solve scheduling problems, chromosome representations are typically categorized into two *paradigms*. One is called *multiple-segment paradigm*, and the other is called *single-segment paradigm*. A few scheduling studies based on these two paradigms have been published [29–31].

These two paradigms are illustrated by referring to a simple scheduling problem, in which there are N jobs to be scheduled in a shop with m parallel machines and each job has only one operation [29–31]. Two types of scheduling decisions are to be made: (1) *job assignment*—assigning each job to one of the m machines, and (2) *job sequence*—determining the processing sequence of the jobs assigned to a particular machine.

To model the two scheduling decisions, a chromosome representation based on the multiple-segment paradigm is as shown in Fig. 3 [31]. In the figure, a chromosome involves two sub-chromosomes that are virtually connected. Each sub-chromosome (also called a segment) is designed to model a scheduling decision. Each element of the first sub-chromosome denotes a job; the sub-chromosome in turn represents a job sequence decision. Each element of the second sub-chromosome denotes a machine, which as a result represents the *job assignment* decision.

In contrast, a chromosome based on the *single-segment paradigm* is as shown in Fig. 4 [29,30]. In the figure, the single-segment chromosome describes a job sequence embedded with two "✽" signs. A "✽" sign denotes a cut-off point; all jobs in the chromosome are thus grouped into three categories (or assigned to the three machines)—the job assignment decision is then made.

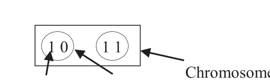

Fig. 2. Allele, gene, and chromosome.

Table 1
Gene representation and gene value in Fig. 2.

Gene representation	Gene value
0 0	0
0 1	0
1 0	1
1 1	1

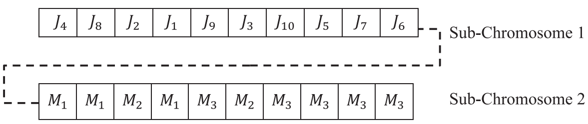

Fig. 3. Multiple-segment chromosome.

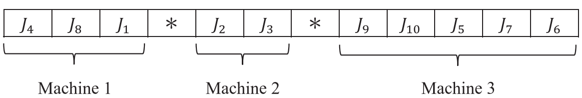

Fig. 4. Single-segment chromosome.

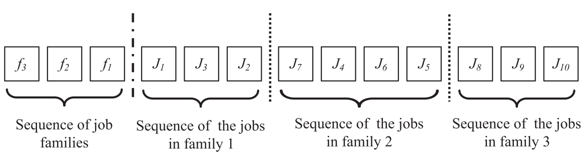

Fig. 5. Traditional chromosome representation (S_{old}).

In turn, the job sequencing decision within each machine can be easily made by referring to the job sequence in the chromosome.

For the PMFS scheduling problem addressed in this research, all prior studies used the same chromosome representation (S_{old}) based on the multiple-segment paradigm [3,20,21–26]. In contrast, we propose a single-segment chromosome representation (S_{new}) and compare the effect of using S_{new} and S_{old} while they are embedded in a particular Tabu-search algorithm.

4. Chromosome representations

This section introduces the two chromosome representations S_{old} and S_{new}. As stated, the PMFS scheduling problem includes two types of decisions—*within-family sequencing* and *among-family sequencing*. Thus, S_{old} and S_{new} must be eligible for accommodating the two types of decisions. In the following, S_{old} and S_{new} are illustrated by referring to a scheduling problem with n jobs (i.e., $J_1, J_{2,\ldots,} J_n$) that have been grouped into k job families (i.e., $f_1, f_{2,\ldots,} f_k$).

4.1. S_{old} representation

To accommodate the two types of sequencing decisions, S_{old} has two distinct features: *clustering* and *multiple-segments*. According to S_{old} representation, a chromosome will have k+1 segments with two clusters. The first cluster involves *only one* segment, which represents the sequence among the k families. The second cluster involves k segments, each of which represents the job sequence within a family.

Fig. 5 illustrates S_{old} representation for a PMFS scheduling problem with 10 jobs and 3 families. The first cluster comprises only one segment, which implies that the sequence among families is $f_3 \rightarrow f_2 \rightarrow f_1$. The second cluster comprises three segments; the first one implies that family f_1 comprises 3 jobs and their processing sequence is $J_1 \rightarrow J_3 \rightarrow J_2$. Accordingly, the second and the third segments respectively represent the job sequence within family f_2 and f_3.

4.2. S_{new} representation

In contrast with S_{old}, S_{new} has two other distinct features: *single-segment* and *decoding mechanism*. A S_{new} chromosome is a *single-segment* which comprises a sequence of jobs. By decoding the chromosome, the two types of scheduling decisions (*within-family sequencing* and *among-family sequencing*) can be obtained.

The decoding mechanism iterates through the chromosome to obtain the sequencing of the families and the jobs. The families are sequenced according to the order of first appearance within the chromosome, while the sequencing of the jobs within the families follows the order of the jobs in the chromosome. To illustrate, Fig. 6 exemplify a scheduling problem with 10 jobs and 3 families. The chromosome indicates that the sequence of the first four jobs is $J_8 \rightarrow J_7 \rightarrow J_4 \rightarrow J_1$ and their corresponding family sequence is $f_3 \rightarrow f_2 \rightarrow f_2 \rightarrow f_1$. This implies that the resulting family sequence is $f_3 \rightarrow f_2 \rightarrow f_1$. By conforming to the job precedence relationships of the chromosome, the job sequence within each family can be easily obtained. That is, the job sequence within family f_3 is $J_8 \rightarrow J_9 \rightarrow J_{10}$, that within$f_2$ is $J_7 \rightarrow J_4 \rightarrow J_6 \rightarrow J_5$, and that within f_1 is $J_1 \rightarrow J_3 \rightarrow J_2$.

5. Tabu search

The pioneer Tabu-search algorithm (called the *TS*) was developed by Glover [13,14]. This research attempts to compare the effect of applying two different representations (S_{old} and S_{new}) to a particular *TS* algorithmic flow. The *TS* algorithms embedded with S_{old} and S_{new} are respectively called *Tabu_*S_{old} and *Tabu_*S_{new}. In the following, we first describe the algorithmic flow of the *TS*, and then present how to apply the *TS* to develop *TS_*S_{old} and *TS_*S_{new}.

5.1. Algorithmic flow of TS

In the *TS* [13,14], a chromosome is a sequence of elements (also called *genes*) and each gene is a positive integer. An example chromosome with five genes is like h=(1, 5, 3, 2, 4). In the algorithmic flow, the *TS* repeatedly uses two modules: (1) *neighborhood generation*; and (2) *tabu_list*. We firstly present the functions of the two modules and proceed to describe the *TS* algorithmic flow.

***Neighborhood generation*:** For a given chromosome h, we could systematically impose a *swap operation* to generate its neighborhood $N(h)$—a set of new chromosomes. An example chromosome is like $h=(a_1, a_2, a_3, a_4, a_5)$. The systematic swap operations are so carried out. On each gene a_i, we iteratively exchange a_i with each of its right-hand side genes. For example, if $h=(a_1, a_2, a_3, a_4, a_5)$, we can generate four new chromosomes by swap a_1 with each of the right-hand side four genes. Accordingly,

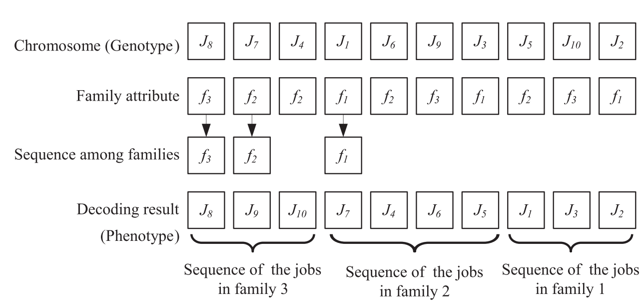

Fig. 6. New chromosome representation (S_{new}).

we can generate three for a_2, two for a_3, and one for a_4. As a result, $N(h)$ in total has $10 = C_2^5$ new chromosomes.

***Tabu_List*:** In a swap operation, the two exchanged genes (say, a_i and a_j) are called a *swap pair* (also called *move* in literature), which is herein denoted by (a_i, a_j). The tabu_list is a set that contains at most q_{size} swap pairs, which are placed in a *sequential* manner. An example tabu_list with $q_{size}=3$ is like {(a_1, a_3), (a_2, a_3), (a_1, a_5)}. The tabu_list dynamically changes its contents whenever a new neighborhood $N(h)$ has been created. If a swap pair (a_i, a_k) is to be placed into the tabu_list, the following two rules must be followed. In the two rules, the tabu_list before placing the swap pair is called the *original_list* and that after placing the swap pair is called the *new_list*.

Rule 1: If (a_i, a_k), the swap pair to be placed into the tabu_list, is also an element in the *original_list*, then the *new_list* is generated by moving the (a_i, a_k) from the present position to the last position. For example, the *original_list* is {(a_1, a_3), (a_2, a_3), (a_1, a_5)} and the swap pair to be placed in is (a_2, a_3). Then, the *new_list* is {(a_1, a_3), (a_1, a_5), (a_2, a_3)}.

Rule 2: If (a_i, a_k) is not an element in the *original_list*, the *new_list* is generated by firstly placing (a_i, a_k) in the last position and then taking away the first element of the *original_list* if the resulting total number of elements is more than q_{size}. That is, if the *original_list* is {(a_1, a_3), (a_2, a_3), (a_1, a_5)} and the swap pair to be placed in is (a_5, a_3). Then, the *new_list* is {(a_2, a_3), (a_1, a_5), (a_5, a_3)}. Alternatively, if the *original_list* is {(a_1, a_3), (a_1, a_5)} which involves only two elements and the swap pair to be placed in is also (a_5, a_3), then the *new_list* is {(a_1, a_3), (a_1, a_5), (a_5, a_3)}.

The algorithmic flow of the *TS*, called ***Tabu_Search*** is presented below.

Procedure *Tabu_Search*

Step 1: Initialization
- generate an initial chromosome h_0; evaluate h_0;
- set $h^*=h_0$; $h^+=h_0$; $T=0$; $Tabu\_list=\phi$;
- /*h^* is the local best solution; h^+ is the global best solution; T is the age of h^+; *Tabu_list* is initially an empty set*/.

Step 2: Generate and sort $N(h^*)$
- generate $N(h^*)$; evaluate each chromosome in $N(h^*)$; set $T=T+1$;
- sort chromosomes in $N(h^*)$ according to their performances;
- name the sorted results as {h_1, h_2,…, h_w}, in which h_i is better than h_{i+1} in terms of performance; name the swap pair that generates h_i from h^* as ω_i.

Step 3: Update h^*, h^+, and Tabu_list
- set $i=1$;/* sequentially justify each h_i in {h_1, h_2, …., h_w}*/;
- while ($i \le w$)/*w is the total number of chromosomes in $N(h^*)$*/.

If $\omega_i \notin Tabu\_list$,
place ω_i in Tabu_list; set $h^*= h_i$;
if h_i is better than h^+, Set $h^+=h_i$, $T=1$; go to Step 4;
elseif $\omega_i \in Tabu\_list$,
if h_i is better than h^*, set $h^*=h_i$; place ω_i in Tabu_list;
if h_i is better than h^+, Set $h^+=h_i$, $T=1$; go to Step 4;
else $i=i+1$
if $i=w+1$, Place ω_1 in Tabu_list; set $h^*=h_1$; go to Step 4;
Endwhile

Step 4: Termination check
- If $T > T_f$, output h^+ and STOP; /*T_f is a predefined large number*/,
- else, go to Step 2.

5.2. Algorithms TS_S_{old} and TS_S_{new}

Procedure ***TS_S_{new}*** is *essentially* the same as ***Tabu_Search*** in algorithmic flow, yet its evaluation of chromosomes is a bit more complicated and needs to be elaborated further. To evaluate a S_{new} chromosome, we have to use the decoding mechanism to obtain its two sequencing decisions, and then evaluate its performance. That is, ***Tabu_Search*** becomes ***TS_S_{new}*** if we elaborate Step 1 as follows. "Evaluate h_0" implies that h_0 (now represented in S_{new}) must be firstly decoded and then evaluated.

As stated, S_{old} uses multiple segments to model a solution. ***TS_S_{old}*** is designed by applying ***Tabu_Search*** to each segment. As a result, ***TS_S_{old}*** has two distinct features: (1) neighborhood generation, (2) multiple tabu_lists. Each feature is respectively explained below.

Neighborhood generation: Consider a problem in which we intend to obtain $N(h^*)$ for a given local best solution h^*. In ***TS_S_{old}***, we generate $N(h^*)$ by applying the swap operation on the segments in a *one-segment-change paradigm*. That is, while we apply the swap operation on a particular segment, we have to keep the other segments of h^* unchanged. For example, if $h^*=s_1–s_2–s_3$ is a three-segment chromosome, then we have to keep s_1 and s_2 unchanged while we apply swap operation on s_3. Suppose each of these three segments has 5 genes. Then we can generate $10 = C_2^5$ new chromosomes for each segment s_i, and the total number of chromosomes in $N(h^*)$ is $30 = 3 \times C_2^5$.

Multiple Tabu_Lists: In ***TS_S_{old}***, a chromosome is composed of *multiple segments*; each segment is designed to have a *unique* tabu_list, and each tabu_list shall be *independently updated*. The *independency* among these multiple tabu_lists is due to that each new chromosomes in $N(h^*)$ is created by applying the swap operation on *only one* particular segment. For example, consider a 3-segment chromosome $h^*=s_1–s_2–s_3$, in which the corresponding tabu_lists of these segments are T_1, T_2, and T_3 respectively. Let h_i be the chromosome which has been identified from $N(h^*)$ for updating h^*. Suppose h_i is created by applying the swap operation on segment s_2. In this case, only tabu_list T_2 shall be updated while we are updating h^* (refer to Step 3 of ***Tabu_Search***).

6. Empirical analysis

Numerical experiments are carried out to compare the two algorithms *Tabu_S_{old}* and *Tabu_S_{new}*, by considering *makespan* as the objective function. Parameters of the two algorithms are so defined: T_f=2000 and $q_{size}=C_2^{n_s} \times p$, where q_{size} denotes the size of tabu_list of *each segment*, p=0.3 and n_s is the total number of genes in a segment. The two algorithms are both coded in C++ programming languages, running on personal computers equipped with AMD Athlon(tm) II*4640 3.0 GHz CPU and 4 GB RAM.

Data sets for the experiments are adopted from prior studies [21]. The data sets are categorized into 30 scenarios, and each scenario includes 30 problem instances. In total, there are 900 (30 × 30) problem instances; each problem instance essentially denotes a unique scheduling problem.

Of the 30 *scenarios*, each one is designated by *X–F–m*, where *X* denotes the type of setup times (LSU, MSU, SSU), *F* is the number of families, and *m* is the number of machines. In addition, SSU denotes small setup times, MSU denotes medium setup times, and LSU denotes large setup times. For example, as shown in Table 1, LSU33 denotes a scenario with large setup times, 3 families, and 3 machines.

Of the 30 problem instances in a scenario, each one is varied by randomly changing the following types of parameters: *the number of jobs per family* (n_f), *processing times*, and *family setup times*.

These parameters are so designed: n_f is randomly generated from a discrete uniform distribution U[1,10]. The processing times at each stage are randomly generated from U[1,10]. Three different cases of setup times were randomly generated, where U[1,20] is used to model SSU, U[1,50] is used to model MSU, and U[1,100] is used to model LSU.

Noticeably, in each problem *instance*, 15 experiments *runs* are carried out. Using a different *random number*, each run generates a different initial solution. In turn, the finally obtained solution in each *run* may be different. Therefore, in each problem *instance*, the average of its 15 experiment runs is taken as the performance of the instance. Furthermore, in each *scenario*, the average of its 30 problem instances is taken as its *ultimate performance measure*. In summary, to compare the two algorithms, we totally carry out 27,000 experiment runs (2 algorithms × 30 scenarios/algorithm × 30 instances/scenario × 15 runs/instance).

Table 2 shows the average experimental results for each of the 30 scenarios. Notation in the table is explained below. The performance (makespan) obtained by *Tabu_S_{old}* is denoted by C_t and that by *Tabu_S_{new}* is denoted by C_n; accordingly, the computation times are respectively denoted by T_t and T_n. The performance difference between $Tabu\_S_{old}$ and $Tabu\_S_{new}$ is denoted by $\gamma=(C_t-C_n)/C_t$. In addition, $N=30$ denotes the total number of instances in a scenario, N_e denotes the number of instances with $\gamma=0$ and N_w denotes the number of instances with $\gamma>0$. In turn, N_w+N_e denotes the number of instances that $Tabu\_S_{new}$ either outperforms or performs equally well as $Tabu\_S_{old}$. The higher is N_w+N_e, the better is $Tabu\_S_{new}$ comparing against $Tabu\_S_{old}$. The table shows that $N_w+N_e=N=30$ in each scenario; and γ ranges from 1.41% to 3.37%, with an average of 2.76%. This indicates that $Tabu\_S_{new}$ apparently outperforms $Tabu\_S_{old}$ in each of the 30 scenarios.

To statistically justify the performance difference between $Tabu\_S_{new}$ and $Tabu\_S_{old}$, a paired *t*-test for the 900 problem instances (30 scenarios × 30 instances/scenario) has been carried out. For each problem instance, the test statistic for modeling the performance difference is defined as $d=(\overline{C}_t-\overline{C}_n)/\overline{C}_t$, where $\overline{C}_t$ and $\overline{C}_n$ respectively denotes the average performance of the 15 runs of the two algorithms. The *t*-value is $t_0=\overline{d}/(S_d/\sqrt{n})$ where $\overline{d}$ is the *mean* and S_d is the *standard deviation* of the 900 problem instances. The obtained *t*-value is $t_0=50.05>t_{0.025,899}=1.96$, which indicates that $Tabu\_S_{new}$ significantly outperforms $Tabu\_S_{old}$. The results demonstrate the importance of representation in enhancing the performance of meta-heuristic algorithms.

Finally, as shown in Table 2, the average of T_n is 26.65 s and that of T_t is 2.14 s. Although T_n appears to be substantially larger than T_t, yet they are both within 3 min in all scenarios. Such little computational efforts are equally acceptable in practice. As stated, $Tabu\_S_{new}$ on average outperforms $Tabu\_S_{old}$ by 2.76% in terms of makespan. This implies that using $Tabu\_S_{new}$ is a good trade-off because the throughput would increase 2.76% at the cost of taking at most 3 min computation.

Table 2
Experimental results of $Tabu\_S_{new}$ and $Tabu\_S_{old}$.

Scenario	Makespan							Computation time (s)	
	N	N_w+N_e	N_e	N_w	C_n	C_t	γ (%)	T_n	T_t
SSU33	30	30	3	27	134.48	137.27	2.04	0.94	0.33
SSU34	30	30	1	29	151.10	153.98	1.92	1.40	0.42
SSU44	30	30	0	30	186.55	191.57	2.65	2.64	0.61
SSU55	30	30	0	30	243.25	250.31	2.81	7.01	1.05
SSU56	30	30	0	30	254.87	262.76	3.04	7.30	1.17
SSU65	30	30	0	30	288.18	296.44	2.79	12.35	1.50
SSU66	30	30	0	30	296.83	305.63	2.88	13.43	1.66
SSU88	30	30	0	30	407.41	419.07	2.77	43.40	3.50
SSU108	30	30	0	30	489.27	504.36	2.99	95.58	5.30
SSU1010	30	30	0	30	530.67	546.58	2.90	128.91	6.83
MSU33	30	30	6	24	160.00	162.41	1.41	0.73	0.28
MSU34	30	30	3	27	184.71	187.60	1.53	1.09	0.38
MSU44	30	30	0	30	238.34	245.51	2.95	2.71	0.60
MSU55	30	30	0	30	309.52	318.11	2.70	6.09	1.03
MSU56	30	30	0	30	319.91	329.14	2.85	6.02	1.08
MSU65	30	30	0	30	366.90	378.18	3.00	10.96	1.43
MSU66	30	30	0	30	385.34	398.26	3.23	12.25	1.64
MSU88	30	30	0	30	521.90	540.04	3.36	36.51	3.30
MSU108	30	30	0	30	640.46	661.39	3.16	86.43	5.42
MSU1010	30	30	0	30	672.82	692.26	2.80	94.50	6.12
LSU33	30	30	4	26	228.09	233.21	2.24	1.06	0.37
LSU34	30	30	4	26	239.06	243.85	2.05	0.88	0.31
LSU44	30	30	0	30	324.89	333.18	2.62	2.41	0.55
LSU55	30	30	0	30	421.59	434.62	3.03	5.70	0.96
LSU56	30	30	0	30	442.82	455.67	2.84	6.84	1.13
LSU65	30	30	0	30	500.06	518.11	3.48	10.63	1.47
LSU66	30	30	0	30	524.49	540.73	3.02	10.93	1.50
LSU88	30	30	0	30	722.17	745.71	3.18	33.00	3.18
LSU108	30	30	0	30	880.50	910.78	3.31	71.70	5.02
LSU1010	30	30	0	30	935.79	968.47	3.37	85.92	6.10
Average	**30**	**30.00**	**0.70**	**29.30**	**400.07**	**412.17**	**2.76**	**26.65**	**2.14**

7. Rationale on the superiority of *Tabu_S_{new}*

This section attempts to explain why $Tabu\_S_{new}$ outperforms $Tabu\_S_{old}$. We start with an extensive observation on the intermediate results of $Tabu\_S_{old}$ for a particular problem instance and found that the *solution-search mechanism* ultimately proceeds in a *loop-search* manner (simply called the *loop feature*). This loop feature leads to that h^+ (the global best solution) cannot be improved any more once the algorithmic flow is trapped into the loop. This observation inspires us to extensively examine whether such a *loop feature* exists in the two algorithms. Further experimental results reveal that $Tabu\_S_{old}$ (compared against $Tabu\_S_{new}$) has a much higher probability of being trapped into a loop. In the following, we firstly summarize the algorithmic flow of the ***Tabu_search*** procedures. Secondly, we describe the method used to examine the loop feature and report the results of the examination. Thirdly, the reason why $Tabu\_S_{old}$ has a much higher probability of being trapped into a loop is analyzed.

7.1. Summary of Tabu search

To facilitate the understanding of the loop feature, we first summarize the algorithmic flow of **Procedure *Tabu_Search*** as stated in Section 5. In essence, the procedure is a *solution evolutionary process*. That is, the proposed solution evolves in a *step-by-step* manner; one such step is called one *evolution-step*. For an *evolution-step i*, we could represent its input by $S_i=(h_i^*,\{T_i^1,\ldots,T_i^k\})$ and its output by $S_{i+1}=(h_{i+1}^*,\{T_{i+1}^1,\ldots,T_{i+1}^k\},)$. In S_i, h_i^* (also called the local best solution) is the *reference chromosome* for creating a set of new solutions $N(h_i^*)$; T_i^q is the tabu_list of *q*-th segment of the chromosome; and the set $\{T_i^1,\ldots,T_i^k\}$ represents all the tabu_list for a multiple segment chromosome.

For an *evolution-step i*, the input/output conversion proceeds as below. First, h_i^* is used to generate $N(h_i^*)$. Then, an appropriate chromosome from $N(h_i^*)$ is selected to be h_{i+1}^*, which in turn could be used to update the global best solution (i.e., obtaining h_{i+1}^+). Notice that h_{i+1}^* is converted from h_i^* by a particular swap pair, which shall be used to update the tabu_lists (i.e., obtaining $\{T_{i+1}^1,\ \ldots,\ T_{i+1}^k\}$).

To record the step-by-step evolution results, we define a notation, called $Track(i,\ i{+}m)=\{S_i,S_{i+1},\ldots,S_{i+m}\}$, which is a set that includes all the input states from evolution-step *i* to *i*+*m*. This implies that the step-by-step results of the whole evolution process can thus be represented by $Track(1,\ N_f)=\{S_1,S_2,\ldots,S_{N_f}\}$, where N_f denotes the total number of neighborhoods that has been generated while the program terminates. In turn, as illustrated in Fig. 7, we can define the *loop feature* as follows: *Track*$(i, i+n)$ is a ***loop*** if $S_i = S_{i+n}$ and $S_i \neq S_{i+k}$ for $1 < k < n$, and n is called the *Loop_size*.

7.2. Examination of loop feature

We examine whether the loop feature exists in the *Tabu_S*$_{old}$ and *Tabu_S*$_{new}$ by a procedure called *Loop_Check* as stated below, in which ***Tabu_Search*** represents either *Tabu_S*$_{old}$ or *Tabu_S*$_{new}$, depending upon the context we are concerned with.

Procedure Loop_Check

Step 1: Carry out the ***Tabu_Search*** process for determining N_f, S_{N_f}
while the ***Tabu_Search*** process terminates (i.e., $h^+_{N_f}$ is obtained),
record N_f, S_{N_f};
set $k = 0$, $i^*_1 = N_f$, $i^*_2 = N_f$.

Step 2: Repeat the ***Tabu_Search*** process for checking loop existence
for each evolution-step $1 \le i \le N_f$
if $(S_i = S_{N_f})$, then/*while a loop seems appear*/
set $k = k+1$, and $i^*_k = i$;/*record the evolution-step*/
endfor
$i^*_{Loop} = i^*_1$;
if $(i^*_1 = N_f)$ then $\kappa = 0$; /*no loop is found*/
if $(i^*_1 < N_f)$ then $\kappa = 1$; /*a loop is found*/
compute *Loop_Size*$= i^*_2 - i^*_1$;
output κ, i^*_1, *Loop_Size*; stop.

In the above procedure, we have $\kappa = 1$ if there exists a loop $(i^*_1 < N_f)$ in the ***Tabu_Search*** process. Once such a loop is found in the *Track*$(1, N_f)$, the *Tabu_Search* process will repeatedly go through the loop. That is, when the ***Tabu_Search*** process reaches at i^*_1 $(i^*_1 < N_f)$, the search process has been trapped into a loop—the size of the loop is *Loop_Size*. This implies that the *effective* search track is at most as long as *Track*$(1, i^*_1 + Loop\_Size)$; and any further search beyond the evolution step $i^*_1 + Loop\_Size$ is in fact repeatedly going around the loop. That is, we cannot obtain any solution better than $h^+_{i^*_1}$.

Due to the loop feature, in order to justify the effectiveness of the search process, we define a notation $\eta = (i^*_1 + Loop\_Size)/N_f$ which is called the *ratio of effective search track*. The higher the value of η, the longer the effective search track, and the better is the search process. Notice that if there is no loop found in the search process $(i^*_1 = N_f)$, we obtain $\eta = 1$ and *Loop_Size*$=0$. In contrast, if there is a loop found $(i^*_1 < N_f)$, then we obtain *Loop_Size* > 0 and $0 < \eta < 1$ in most cases. In very rare cases, we may obtain $\eta = 1$ and *Loop_Size* > 0 while $i^*_2 = N_f$.

As stated, in the experiment, there are 30 scenarios; each scenario has 30 instances; and each instance has 15 replicates. In other words, each scenario involves 450 (30×15) experiment runs in total; and each run shall yield a κ value and a η value; let $\bar{\kappa}$ and $\bar{\eta}$ respectively represent the averages of the 450 runs in a scenario. The values of $\bar{\kappa}$ and $\bar{\eta}$ both range from 0 to 1. Herein, $\bar{\kappa}$ is defined as the *loop frequency indicator*. The higher the $\bar{\kappa}$ value, the more frequently the loop features would appear in a scenario, and the less effective is the search algorithm. In addition, $\bar{\eta}$ is called the *average ratio of effective search track*. The higher the value of $\bar{\eta}$, the longer the effective search track, and the more effective is the search algorithm.

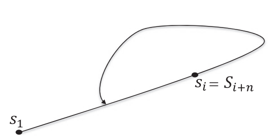

Fig. 7. The loop feature.

Table 3 shows the comparison of the two algorithms *Tabu_S*$_{new}$ and *Tabu_s*$_{old}$ in terms of $\bar{\kappa}$ and $\bar{\eta}$. The table indicates that *Tabu_S*$_{old}$ has a very high $\bar{\kappa}$ value (i.e., 85.46% on average); that is, about 85% *Tabu_S*$_{old}$ experiment runs would be trapped into a loop. In contrast, *Tabu_S*$_{new}$ has a very low $\bar{\kappa}$ value (i.e., 0.38% on average); this implies that *Tabu_S*$_{new}$ experiment runs shall be rarely trapped into a loop.

In addition, *Tabu_S*$_{old}$ has a low $\bar{\eta}$ value (i.e., 24.93% on average). This implies that the search track that *Tabu_S*$_{old}$ goes through is only 24.93% effective; and the remaining 75.07% search track is in fact ineffective because the search process now has been trapped into a loop. In contrast, *Tabu_S*$_{new}$ has a very high $\bar{\eta}$ value (i.e., 99.75% on average); this implies that *Tabu_S*$_{new}$ experiment runs are very rarely trapped into a loop.

In summary, the reason why *Tabu_S*$_{new}$ outperforms *Tabu_S*$_{old}$ is due to the loop feature, the degree of which are measured by two indicators $\bar{\kappa}$and $\bar{\eta}$ The $\bar{\kappa}$ value reveals that *Tabu_S*$_{old}$ has a much higher probability of being trapped into a loop; and the $\bar{\eta}$ value indicates that the effective search track of *Tabu_S*$_{old}$ tend to be much shorter, due to being trapped into a loop.

7.3. Effect of S_{new} and S_{old} on loop feature

This section attempts to explain why *Tabu_S*$_{old}$ (compared against *Tabu_S*$_{new}$) has a much higher probability of being trapped into a loop. As stated, *Tabu_S*$_{old}$ and *Tabu_S*$_{new}$ both evolve in a *step-by-step* manner. For an *evolution-step i*, its input state is modeled by $S_i = (h^*_i, \{T^1_i, \ldots, T^k_i\})$, where h^*_i is the *reference chromosome* for creating a set of new solutions $N(h^*_i)$ and T^q_i is the tabu_list of q-th segment of the chromosome; in turn the set $\{T^1_i, \ldots, T^k_i\}$ represents all the tabu_list for a k-segment chromosome. Herein, we define $\Psi(S)$ as the modeling space of S_i. That is, $\Psi(S)$ is a set that contains all possible instances generated by freely varying each component in S_i.

Being trapped into a loop indicates that we could find an $S_i = S_j$, where $i \neq j$. This implies that, for a ***Tabu_Search*** algorithm, the larger is $\Psi(S)$, the lower is the probability of getting an $S_i = S_j$; in turn the algorithm would have a lower probability of being trapped into a loop. Define $\Psi_{old}(S)$ and $\Psi_{new}(S)$ respectively as the modeling space in *Tabu_S*$_{old}$ and *Tabu_S*$_{new}$.

Taking the scheduling problem in Fig. 5as an example, we analyze the complexity of $\Psi_{old}(S)$ and $\Psi_{new}(S)$ as below. For $\Psi_{old}(S)$ in this case, there are four segments (s_1–s_2–s_3–s_4). In segment s_3, there are 4 genes, which leads to $q_{size} = C^4_2 \times 0.3 = [1.8] = 2$. The possible number of *segment instances* is 4!. The possible number of tabu_list instances is 37 as explained below. In segment s_3, the possible number of swap pair is $C^4_2 = 6$, and there are two *slots* in the tabu_list $(q_{size} = 2)$. Each slot can be filled in either by a swap pair or nothing. Then the possible number of tabu_list instances is $(6 \times 5) + (6 \times 1) + (1 \times 1) = P^6_2 + P^6_1 + P^6_0$, where the first term denotes that the two slots are both filled, the second term denotes that only the first slot is filled and the second slot is empty, and the third term denotes that both the two slots are empty.

Following the above example procedure, for a segment s_i with n_i genes and $q_{size} = q_i$, we could obtain the modeling space of the segment as follows. The possible number of *segment instances* is $n_i!$. The possible number of swap pair is $C^{n_i}_2$, and there are q_i *slots* in the tabu_list. Then the possible number of *tabu_list instances* is $P^{C^{n_i}_2}_{q_i} + P^{C^{n_i}_2}_{q_i - 1} + \ldots + P^{C^{n_i}_2}_0 = \sum^{q_i}_{j=0} P^{C^{n_i}_2}_j$; in turn, the modeling space of this segment is $n_i! \times \sum^{q_i}_{j=0} P^{C^{n_i}_2}_j$.

Table 3
Experimental results of the loop-feature examination.

Scenario	$\overline{\kappa}$ (%)		$\overline{\eta}$ (%)	
	$\overline{\kappa}_{new}$	$\overline{\kappa}_{old}$	$\overline{\eta}_{new}$	$\overline{\eta}_{old}$
SSU0	4.14	86.00	96.40	24.92
SSU1	1.61	80.92	99.46	30.11
SSU2	0.00	85.06	100.00	25.05
SSU3	0.00	81.38	100.00	28.78
SSU4	0.00	87.59	100.00	23.92
SSU5	0.00	82.99	100.00	27.79
SSU6	0.00	72.87	100.00	36.93
SSU7	0.00	71.03	100.00	41.50
SSU8	0.00	83.45	100.00	26.96
SSU9	0.00	78.39	100.00	32.61
MSU0	0.46	87.11	99.94	21.36
MSU1	0.00	81.84	99.65	26.19
MSU2	0.00	93.10	100.00	18.67
MSU3	0.00	87.13	100.00	22.50
MSU4	0.00	85.98	100.00	25.33
MSU5	0.00	94.94	100.00	15.41
MSU6	0.00	81.61	100.00	28.76
MSU7	0.00	84.37	100.00	27.51
MSU8	0.00	79.31	100.00	29.58
MSU9	0.00	86.67	100.00	24.98
LSU0	2.99	88.05	98.78	21.41
LSU1	0.22	91.49	99.80	18.57
LSU2	2.07	91.26	98.53	17.52
LSU3	0.00	88.51	100.00	21.17
LSU4	0.00	84.83	100.00	26.18
LSU5	0.00	90.80	100.00	20.26
LSU6	0.00	94.02	100.00	14.18
LSU7	0.00	91.95	100.00	19.53
LSU8	0.00	82.99	100.00	27.04
LSU9	0.00	88.05	100.00	23.02
Average	**0.38**	**85.46**	**99.75**	**24.93**

Table 4
Experiments with different p values for $Tabu\_S_{new}$ and $Tabu\_S_{old}$.

Average	**Makespan**							**Computation time (s)**	
***p* Value**	$\boldsymbol{N}$	$\boldsymbol{N_w+N_e}$	$\boldsymbol{N_e}$	$\boldsymbol{N_w}$	$\boldsymbol{C_n}$	$\boldsymbol{C_t}$	γ (%)	$\boldsymbol{T_n}$	$\boldsymbol{T_t}$
0.3	**30**	**30.00**	**0.70**	**29.30**	**400.07**	**412.17**	**2.76**	**26.65**	**2.14**
0.5	**30**	**30.00**	**1.17**	**28.83**	**400.00**	**411.51**	**2.60**	**26.91**	**2.18**
0.7	**30**	**29.97**	**1.37**	**28.60**	**399.86**	**410.49**	**2.37**	**27.77**	**2.26**
0.9	**30**	**29.97**	**1.90**	**28.07**	**398.37**	**408.98**	**2.27**	**28.82**	**2.27**

Now consider a chromosome with k segments; we could accordingly obtain that the number of elements in its $\Psi(S)$ is $\prod_{i=1}^{k}(n_i! \times \sum_{j=0}^{q_i} P_j^{C_2^{n_i}})$. Following this formula, for the scheduling problem in Fig. 1, the number of elements in $\Psi_{old}(S)$ is 5.6×10^6 and that in $\Psi_{new}(S)$ is 5.4×10^{28}. This indicates that $\Psi_{old}(S)$ is much smaller than $\Psi_{new}(S)$; as a result, $Tabu\_S_{old}$ tends to have a higher probability of being trapped into a loop.

In summary, the reason why $Tabu\_S_{new}$ outperforms $Tabu\_S_{old}$ may be due to that $\Psi_{new}(S)$ has a higher degree of freedom than $\Psi_{old}(S)$. This implies that increasing the degree of freedom of $\Psi(S)$ might improve the solution quality. We justify this hypothesis by increasing the tabu_list size ($q_{size}=C_2^{n_i}\times p$) by setting p=0.3, 0.5, 0.7, 0.9; and comprehensively carry out the numerical experiments for each p value. As shown in Table 4, experiments results reveal two important findings. Firstly, $Tabu\_S_{new}$ keeps outperforming $Tabu\_S_{old}$ for each p value. Secondly, $Tabu\_S_{new}$ and $Tabu\_S_{old}$ both improve their performances while we increase p value. These two findings essentially support our hypothesis—a $\Psi(S)$ with a higher degree of freedom tends to yield a better solution.

8. $GA\_Tabu\_S_{new}$ and $GA\_Tabu\_S_{old}$

As stated, this research has two objectives. The first objective is to compare $Tabu\_S_{new}$ and $Tabu\_S_{old}$. The second objective is to develop a meta-heuristic algorithm that outperforms the state-of-the-art algorithms in solving the PMFS problem. To fulfill the second objective, we develop two algorithms $GA\_Tabu\_S_{new}$ and $GA\_Tabu\_S_{old}$, and compare their solution quality with the latest benchmark algorithms [3,20].

Adopting S_{new} as the solution representation scheme, the $GA\_Tabu\_S_{new}$ algorithm is a two-stage evolutionary process, a mixture of global search and local search. The first stage is the use of GA_S_{new} for carrying out a global search. The second stage is the use of $Tabu\_S_{new}$ for carrying out a local search, by taking the solution obtained from GA_S_{new} as the input (i.e., the initial solution of $Tabu\_S_{new}$). By contrast, $GA\_Tabu\_S_{old}$ is a mixture of GA_S_{old} and $Tabu\_S_{old}$, which also adopts the two-stage evolutionary process but uses S_{old} as the solution representation scheme.

The parameters of $GA\_Tabu\_S_{new}$ and $GA\_Tabu\_S_{old}$ are both set as follows: P_{size}=1000, p_c=0.95, p_m=0.10, T_f^{GA}=3,000,000, T_f^{TS}=2000 and $q_{size}=C_2^{n_s}\times 0.3$, where T_f^{G} is the termination condition of GA evolution and T_f^{TS} is the termination condition of TS evolution.

Numerical experiments for comparing the two algorithms ($GA\_Tabu\_S_{new}$ and $GA\_Tabu\_S_{old}$) with other benchmark algorithms are carried out. Table 5 compares the solution quality, with their computation times shown in Table 6. Table 5 indicates that $GA\_Tabu\_S_{new}$ outperforms all the other algorithms; this finding is statistically significant justified from paired t-tests (Table 7). In addition, the computation times required for $GA\_Tabu\_S_{new}$ in each scenario is less than 3 min (Table 6), which is computationally efficient from the perspective of practical applications.

Experimental results indicate that S_{new} appears to be superior to S_{old} while they are embedded in a *particular* meta-heuristic algorithm. That is, X_S_{new} is superior to X_S_{old} where X denotes a particular meta-heuristic algorithm such as *GA*, *ACO*, *Tabu*, and *GA_Tabu*. These findings, supported by their differences in makespan as shown in Table 5, have been justified to be statistically significant by paired t-tests (Table 8).

9. Conclusions

This research, investigating the application of meta-heuristic algorithms to solve the PMFS scheduling problem, has two objectives. First, we attempt to compare the effect of using two different solution representations (S_{new} *and* S_{old}) while applying the *Tabu* algorithm. Second, we attempt to develop a meta-heuristic algorithm that outperforms the state-of-the-art meta-heuristic algorithms for solving the PMFS problem.

For the first objective, experimental results indicate that $Tabu\_S_{new}$ outperforms $Tabu\_S_{old}$ in terms of solution quality, with practically acceptable computational efforts (requiring only a few minutes). The reason why $Tabu\_S_{old}$ is inferior is due to that it tends to be trapped into a loop. The loop feature is due to that S_{old} in nature has a relatively lower degree of freedom than S_{new} in modeling a Tabu neighborhood. This in turn increases the probability of visiting a state that have been searched and leads to a loop search.

For the second objective, two meta-heuristic algorithms ($GA\_Tabu\_S_{new}$ and $GA\_Tabu\_S_{old}$) are proposed. Experimental results indicate that $GA\_Tabu\_S_{new}$ outperforms all the other meta-heuristic algorithms to date which includes the state-of-the art algorithms GA_S_{old} [3] and GA_S_{new} [20]. In addition, results of paired t-tests reveal that X_S_{new} is superior to X_S_{old} where X denotes one of the following four: *GA*, *Tabu*, *ACO*, and *GA_Tabu*.

Table 5
Experiments for comparing algorithms in terms of makespan.

Scenario	Makespan							
	Tabu		*GA*		*ACO*		*GA_Tabu*	
	Tabu_S_{new}	Tabu_S_{old}	GA_S_{new}	GA_S_{old}	ACO_S_{new}	ACO_S_{old}	GA_Tabu_S_{new}	GA_Tabu_S_{old}
SSU33	134.48	137.27	134.47	134.47	134.96	136.27	134.47	134.47
SSU34	151.1	153.98	150.98	150.99	152.56	154.46	150.98	151
SSU44	186.55	191.57	185.64	185.64	187.63	190.84	185.63	185.64
SSU55	243.25	250.31	241.94	242.11	246.79	250.53	241.9	242.09
SSU56	254.87	262.76	253.36	253.71	259.57	262.97	253.36	253.69
SSU65	288.18	296.44	285.78	285.96	292.51	297.35	285.77	285.95
SSU66	296.83	305.63	294.9	295.14	302.18	308.71	294.87	295.12
SSU88	407.41	419.07	402.6	403.14	418.97	424.28	402.46	403.12
SSU108	489.27	504.36	481.76	481.94	506.84	512.1	481.61	481.92
SSU1010	530.67	546.58	521.9	522.45	551.92	554.87	521.73	522.42
MSU33	160	162.41	160	160	160.76	162.74	160	160
MSU34	184.71	187.6	184.68	184.69	185.23	187.42	184.67	184.7
MSU44	238.34	245.51	237.6	237.61	239.03	242.31	237.6	237.6
MSU55	309.52	318.11	306.09	306.16	309.46	315.96	306.08	306.15
MSU56	319.91	329.14	317.47	317.68	321.72	329.32	317.47	317.68
MSU65	366.9	378.18	362.63	362.66	367.47	375.24	362.63	362.66
MSU66	385.34	398.26	380.02	380.07	386.52	396.05	380.02	380.06
MSU88	521.9	540.04	510.39	510.44	529.09	546.14	510.21	510.45
MSU108	640.46	661.39	623.95	623.51	655.03	667.53	623.88	623.49
MSU1010	672.82	692.26	655.17	655.52	687.61	698.2	655	655.48
LSU33	228.09	233.21	228.03	228.03	228.67	229.98	228.03	228.03
LSU34	239.06	243.85	239	239	239.38	241.19	239	239
LSU44	324.89	333.18	323.44	323.43	324.13	327.27	323.44	323.43
LSU55	421.59	434.62	415.97	415.97	419.19	426.5	415.97	415.97
LSU56	442.82	455.67	436.97	437.17	440.7	449.42	436.98	437.16
LSU65	500.06	518.11	491.92	492.02	496.02	507.07	491.92	492.02
LSU66	524.49	540.73	514.32	514.34	518.88	531.96	514.32	514.34
LSU88	722.17	745.71	701.63	701.11	715.14	736.53	701.62	701.11
LSU108	880.5	910.78	847.62	847.36	883.67	905.5	847.61	847.33
LSU1010	935.79	968.47	907.13	908.67	947.02	972.93	907.14	908.63
Average	**400.07**	**412.17**	**393.25**	**393.37**	**403.62**	**411.39**	**393.21**	**393.36**

Table 6
Experiments for comparing algorithms in terms of computation time.

Scenario	Computation time (s)							
	Tabu		*GA*		*ACO*		*GA_Tabu*	
	Tabu_S_{new}	Tabu_S_{old}	GA_S_{new}	GA_S_{old}	ACO_S_{new}	ACO_S_{old}	GA_Tabu_S_{new}	GA_Tabu_S_{old}
SSU33	0.94	0.33	14.88	18.54	3.35	0.64	11.75	14.67
SSU34	1.4	0.42	16.84	20.5	5.15	1.28	13.72	16.35
SSU44	2.64	0.61	21.17	24.56	9.77	2.14	17.97	19.68
SSU55	7.01	1.05	30.63	32.3	19.49	4.69	28.42	26.49
SSU56	7.3	1.17	31.95	34.81	21.03	6.7	29.58	28.55
SSU65	12.35	1.5	36.41	37.27	31.09	7.36	36.8	31.49
SSU66	13.43	1.66	40.16	40.03	31.67	7.72	40.68	33.57
SSU88	43.4	3.5	67.49	58.19	57.37	18.29	80.4	50.85
SSU108	95.58	5.3	95.61	72.24	100.11	25.48	134.05	65.27
SSU1010	128.91	6.83	107.94	83.84	98.01	29.51	161.38	76.64
MSU33	0.73	0.28	13.75	17.91	3.38	0.51	10.83	14.09
MSU34	1.09	0.38	15.7	19.59	5.13	0.97	12.69	15.66
MSU44	2.71	0.6	21.01	24.23	12.76	2.16	17.87	19.54
MSU55	6.09	1.03	29.64	31.42	31.04	5.66	27.37	25.83
MSU56	6.02	1.08	30.04	32.98	30.41	4.9	27.61	27.08
MSU65	10.96	1.43	35.92	36.68	49.92	7.51	35.84	30.79
MSU66	12.25	1.64	38.55	39.33	49.08	8.39	38.76	33.26
MSU88	36.51	3.3	65.15	56.55	102.58	20.49	78.2	50.2
MSU108	86.43	5.42	89.16	70.74	175.06	39.82	128.69	64.35
MSU1010	94.5	6.12	99.95	79.66	164.46	40.3	143.5	72.21
LSU33	1.06	0.37	15.44	18.79	4.98	0.8	12.39	15.1
LSU34	0.88	0.31	14.94	18.96	5.5	0.83	11.91	15.05
LSU44	2.41	0.55	19.99	23.81	16.36	1.95	17.02	19.06
LSU55	5.7	0.96	28	30.95	48.32	4.75	25.62	25.28
LSU56	6.84	1.13	30.72	33.24	50.98	5.8	28.61	27.44
LSU65	10.63	1.47	35.95	36.48	92.34	6.75	35.94	30.7
LSU66	10.93	1.5	36.37	38.44	87.37	8.96	36.47	32.5
LSU88	33	3.18	60.78	54.9	180.73	22.54	72.55	48.51
LSU108	71.7	5.02	85.85	68.94	328.03	64.14	121.7	62.46
LSU1010	85.92	6.1	101.14	78.04	295.34	61.76	138.82	71.05
Average	**26.65**	**2.14**	**44.37**	**41.13**	**70.36**	**13.76**	**52.57**	**35.46**

Table 7
Paired *t*-tests for supporting that $GA\_Tabu\_S_{new}$ outperforms the other algorithms.

Paired *t*-test	$GA\_Tabu\_S_{new}$ vs. other algorithms						
Algorithm	$Tabu\_S_{old}$	$Tabu\_S_{new}$	GA_S_{old}	GA_S_{new}	ACO_S_{old}	ACO_S_{new}	$GA\_Tabu\_S_{old}$
t-Value	**53.91**	**27.75**	**3.90**	**5.71**	**44.06**	**31.79**	**3.66**

Table 8
Paired *t*-test for supporting that S_{new} outperforms S_{old} for each of the following four meta-heuristic algorithms.

Paired *t*-test	S_{new} vs. S_{old}			
Algorithm	*Tabu*	*GA*	*ACO*	*GA_Tabu*
t-Value	**50.05**	**3.16**	**30.88**	**3.66**

This research highlights the importance of developing novice solution representations in the application of meta-heuristic algorithms. This idea can be extended to investigate other scheduling problems or other space search problems that have been solved by meta-heuristic algorithms.

Acknowledgments

This work is financially supported by a research contract NSC 99-2221-E-009-110-MY3.

References

[1] Rothlauf F, Goldberg DE, Heinzl A. Network random keys—a tree network representation scheme for genetic and evolutionary algorithms. Evolutionary Computation 2002;10(1):75–97.
[2] Rothlauf F, Goldberg DE. Redundant representations in evolutionary computation. Evolutionary Computation 2003;11(4):381–415.
[3] Lin SW, Ying KC, Lee ZJ. Metaheuristics for scheduling a non-permutation flowline manufacturing cell with sequence dependent family setup times. Computers & Operations Research 2009;36:1110–21.
[4] Carotenuto P, Giordani S, Ricciardelli S, Rismondo S. A Tabu search approach for scheduling hazmat shipments. Computers & Operations Research 2007;34:1328–50.
[5] Holland JH. Adaptation in neural and artificial systems. Ann Arbor, Michigan: University of Michigan Press; 1975.
[6] Burak E, Sandra DE, Jain P. A Tabu search algorithm for the flowshop scheduling problem with changing neighborhoods. Computers & Industrial Engineering 2008;54:1–11.
[7] Buscher U, Shen L. An integrated Tabu search algorithm for the lot streaming problem in job shops. European Journal of Operational Research 2009;199:385–99.
[8] Wen UP, Huang AD. A simple Tabu search method to solve the mixed-integer linear bilevel programming problem. European Journal of Operational Research 1996;88:563–71.
[9] Kim YK, Kim JY, Kang SS. A Tabu search approach for designing a non-hierarchical video-on-demand network architecture. Computers & Industrial Engineering 1997;33(3–4):837–40.
[10] Lokketangen A, Glover F. Solving zero–one mixed integer programming problems using Tabu search. European Journal of Operational Research 1998;106:624–58.
[11] Cordeau JF, Maischberger M. A parallel iterated Tabu search heuristic for vehicle routing problems. Computers & Operations Research 2012;39:2033–55.
[12] Demir L, Tunalı S, Eliiyi DT. An adaptive Tabu search approach for buffer allocation problem in unreliable non-homogenous production lines. Computers & Operations Research 2012;39:1477–86.
[13] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165:479–94.
[14] Hejazi SR, Saghafian S. Flowshop-scheduling problems with makespan criterion: a review. International Journal of Production Research 2005;43 (14):2895–929.
[15] Glover F. Tabu search Part I. ORSA Journal of Computing 1989;1:190–206.
[16] Glover F. Tabu search Part II. ORSA Journal of Computing 1990;2:4–32.
[17] Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 2002;1(1):53–66.
[18] Kirkpatrick S, Gelatt Jr. CD, Vecchi MP. Optimization by simulated annealing. Science 1983;202:671–80.
[19] Goldberg DE. The design of innovation: lessons from and for competent genetic algorithms. Boston, MA: Kluwer Academic Publishers; 2002.
[20] Chen CF, Wu MC, Li YH, Tai PH, Chiou CW. A comparison of two chromosome representation schemes used in solving a family-based scheduling problem. Robotics and Computer-Integrated Manufacturing 2012;29(3):21–30.
[21] Schaller JE, Gupta JND, Vakharia AJ. Scheduling a flowline manufacturing cell with sequence dependent family setup times. European Journal of Operational Research 2000;125(2):324–39.
[22] Gupta JND, Stafford EF. Flowshop scheduling research after five decades. European Journal of Operational Research 2006;169:699–711.
[23] Hendizadeh SH, Faramarzi H, Mansouric SA, Gupta JND, ElMekkawy TY. Meta-heuristics for scheduling a flowline manufacturing cell with sequence dependent family setup times. International Journal of Production Economics 2008;111(2):593–605.
[24] Ying KC, Gupta JND, Lin SW, Lee ZJ. Permutation and nonpermutation schedules for the flowline manufacturing cell with sequence dependent family setups. International Journal of Production Research 2009;48 (8):2169–84.
[25] Franca PM, Gupta JND, Mendes AS. Evolutionary algorithms for scheduling a flowshop manufacturing cell with sequence dependent family setups. Computers & Industrial Engineering 2005;48(3):491–506.
[26] Lin SW, Gupta JND, Ying KC, Lee ZJ. Using simulated annealing to schedule a flowshop manufacturing cell with sequence dependent family setup times. International Journal of Production Research 2009;47(12):3205–17.
[27] Bhushan S, Karimi IA. Heuristic algorithms for scheduling an automated wet-etch station. Computers & Chemical Engineering 2004;28(3):363–79.
[28] Castro PM, Aguirre AM, Zeballos LJ, Méndez CA. Hybrid mathematical programming discrete-event simulation approach for large-scale scheduling problems. Industrial & Engineering Chemistry Research 2011;50(18): 10665–10680.
[29] Chan FTS, Choy KL, Bibhushan. A genetic algorithm-based scheduler for multiproduct parallel machine sheet metal job shop. Expert Systems with Applications 2011;38:8703–15.
[30] Tavakkoli-Moghaddam R, Taheri F, Bazzazi M, Izadi M, Sassani F. Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. Computers & Operations Research 2009;36:3224–30.
[31] M'Hallah R, Al-Khamis T. Minimising total weighted earliness and tardiness on parallel machines using a hybrid heuristic. International Journal of Production Research 2011;1:1–26.
[32] Palmer CC. An approach to a problem in network design using genetic algorithms. (Unpublished PhD thesis), Polytechnic University, Troy, NY; 1994.