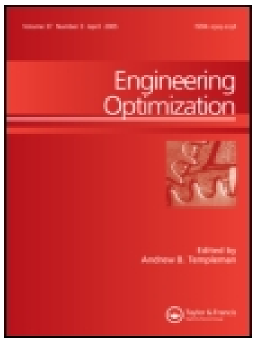

Engineering Optimization

ISSN: 0305-215X (Print) 1029-0273 (Online) Journal homepage: https://www.tandfonline.com/loi/geno20

A hybrid dynamic harmony search algorithm for identical parallel machines scheduling

Jing Chen , Quan-Ke Pan , Ling Wang & Jun-Qing Li

To cite this article: Jing Chen , Quan-Ke Pan , Ling Wang & Jun-Qing Li (2012) A hybrid dynamic harmony search algorithm for identical parallel machines scheduling, Engineering Optimization, 44:2, 209-224, DOI: 10.1080/0305215X.2011.576759

To link to this article: https://doi.org/10.1080/0305215X.2011.576759

Published online: 27 Jun 2011.

Engineering Optimization
Vol. 44, No. 2, February 2012, 209–224

A hybrid dynamic harmony search algorithm for identical parallel machines scheduling

Jing Chen[a], Quan-Ke Pan[b]*, Ling Wang[c] and Jun-Qing Li[a]

[a]*College of Computer Science, Liaocheng University, Liaocheng, People's Republic of China;* [b]*State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan, 430074, People's Republic of China;* [c]*TNList, Department of Automation, Tsinghua University, Beijing, People's Republic of China*

(*Received 30 May 2010; final version received 3 March 2011*)

In this article, a dynamic harmony search (DHS) algorithm is proposed for the identical parallel machines scheduling problem with the objective to minimize makespan. First, an encoding scheme based on a list scheduling rule is developed to convert the continuous harmony vectors to discrete job assignments. Second, the whole harmony memory (HM) is divided into multiple small-sized sub-HMs, and each sub-HM performs evolution independently and exchanges information with others periodically by using a regrouping schedule. Third, a novel improvisation process is applied to generate a new harmony by making use of the information of harmony vectors in each sub-HM. Moreover, a local search strategy is presented and incorporated into the DHS algorithm to find promising solutions. Simulation results show that the hybrid DHS (DHS_LS) is very competitive in comparison to its competitors in terms of mean performance and average computational time.

Keywords: parallel machines scheduling; makespan; harmony search algorithm; variable neighbourhood search; local search

1. Introduction

This article considers the identical parallel machines scheduling (IPMS) problem with the objective to minimize makespan (the maximum completion time), which is one of the classical combinatorial optimization problems with considerable attention in the past decades and has been proved to be NP-hard by Garey *et al.* (1979). To solve IPMS problem, heuristics and meta-heuristics have been commonly used. Based on the idea of list scheduling (LS), Graham *et al.* (1969) proposed the LPT (longest processing time) method. The jobs are first sorted in decreasing order according to their processing times. Starting with the first job, all jobs are then successively assigned to the least loaded machine. The LPT heuristic has received extensive attention and has been shown to perform well for the makespan single criteria problem. Inspired by the similarity between the bin-packing problem and the IPMS problem, Coffman *et al.* (1978) proposed another

*Corresponding author. Email: panquanke@gmail.com

ISSN 0305-215X print/ISSN 1029-0273 online
© 2012 Taylor & Francis
http://dx.doi.org/10.1080/0305215X.2011.576759
http://www.tandfonline.com

classical heuristic MULTIFIT, which tried to find the minimal capacity by utilizing a bin-packing method such that all n jobs fit into the m bins. Its worst-case performance has been proved to be better than that of the LPT method. Lee and Massey (1988) combined the above two heuristics by utilizing the result of LPT as the incumbent and then applying MULTIFIT with fewer iterations. It has been shown to have a worst case that is never worse than MULTIFIT and the performance is as good as the best of either of LPT and MULTIFIT. Ghomi and Ghazvini (1998) used a pairwise interchange (PI) method to generate near-optimal solutions, which can also be applied to the problems of non-identical processors and non-simultaneous arrivals. Gupta *et al.* (2001) presented a LISTFIT heuristic based on bin-packing and list scheduling, which outperformed the LPT, MULTIFIT, and their hybridization.

In recent years, modern meta-heuristics have been increasingly applied to the IPMS problem (Liu and Cheng 1999, Costa *et al.* 2002, Lee *et al.* 2006, Sevkli and Uysal 2009, Kashan and Karimi 2009). Liu and Cheng (1999) proposed a genetic algorithm (GA) by using a machine-based encoding method, and they found that the GA was suitable for larger scale problems when compared with the LPT (Graham 1969) and simulated annealing (SA) methods. Costa *et al.* (2002) presented an immune-based approach based on the way the vertebrate immune system works. The performance of this method has been improved by maintaining a diverse set of near-optimal solutions along the search. Using both exchange and insert neighbourhood structures, Sevkli and Uysal (2009) proposed a modified variable neighbourhood search (VNS) algorithm and demonstrated that their algorithm outperformed GA (Liu and Cheng 1999) and LPT (Graham 1969). Lee *et al.* (2006) proposed a SA algorithm by using the LPT sequence as its initial sequence. Computational results showed that it worked better than both the LISTFIT (Gupta *et al.* 2001) and the PI (Fatemi *et al.* 1998) methods and was very efficient for large-sized problems. Kashan and Karimi (2009) introduced a discrete particle swarm optimization (DPSO), where the velocity and position equations were changed to be suitable for this discrete optimization problem. The method was found by the authors to be more competitive than SA after hybridizing it with a local search scheme. Due to both the SA (Lee *et al.* 2006) and the DPSO (Kashan and Karimi 2009) methods having good performances in the literature, they will be implemented in this article as comparator algorithms to calibrate the proposed method.

More recently, a new meta-heuristic, named harmony search (HS), has been proposed by Geem *et al.* (2001). Its basic idea was inspired by the improvisation process of musicians. In HS, the process that the music players improvise the pitches for finding the best harmony is similar to the process of searching for the optimal solution in the solution space. Due to its simplicity, few parameters, and easy implementation, HS has received much attention and has been applied to a wide range of optimization problems (Mahdavi *et al.* 2007, Cheng *et al.* 2008, Omran and Mahdavi 2008, Lee and Geem 2005, Geem *et al.* 2005). However, the applications of HS on combinatorial optimization problems are still limited. In this article, a dynamic HS (DHS) algorithm is presented for the IPMS problem. In particular, a list scheduling (LS)-based rule is used to convert continuous harmony vectors to discrete machine numbers. To balance population diversity and fast convergence rate, the whole harmony memory (HM) is divided into multiple small-sized sub-HMs, which are regrouped periodically by using a regrouping scheme, and information is then exchanged dynamically. Moreover, a new improvisation scheme is presented to generate candidate solutions so as to take full advantage of useful information of harmonies in each sub-HM. Finally, a variable neighbourhood search is designed to perform an elaborate local search. The performances of the proposed algorithms are compared with the exiting methods by using a large number of problems, and the results show that the hybrid DHS (DHS_LS) outperforms its competitors in terms of mean performance and average computational time.

The rest of this article is organized as follows. In Section 2, a brief description of IPMS problem is given. In Section 3, the basic HS algorithm and the presented dynamic HS algorithm are described. Then, the encoding strategy, population initialization and VNS-based local search are given in detail in Section 4. The computational results and comparisons are presented in Section 5. Finally, the article is concluded in Section 6.

2. Problem description

The IPMS problem can be described as follows: suppose there is a set of n jobs $J = \{J_1, J_2, \ldots, J_n\}$ and a set of m identical parallel machines $M = \{M_1, M_2, \ldots, M_m\}$. Each job becomes available for processing at the start time, and requires only one operation, which can be processed on one of the m machines without any interruption. The processing time of job J_i is denoted as p_i. There is no priority and no precedence relationships between jobs. Moreover, each machine can process only one job at a time. The goal is to find an optimal schedule where the given n jobs are assigned to m identical machines properly so that the completion time of the most loaded machine, *i.e.* the makespan, is the lowest possible. Let $S = \{S_1, S_2, \ldots, S_m\}$ denotes a schedule, where S_j represents the subset of jobs assigned to machines M_j with completion time $C_j = \sum_{i\in S_j} p_i$. A mathematical formulation of this problem is given as follows:

$$\begin{aligned}
\min \quad & C_{max} = \min\left(\max_{1\le j\le m} C_j\right) = \min\left(\max_{1\le j\le m} \sum_{i\in S_j} p_i\right) \\
\text{s.t.} \quad & C_{max} \sum_{i=1}^{n} p_i \cdot x_{ij} \ge 0 \quad j = 1, 2, \ldots, m \\
& \sum_{j=1}^{m} x_{ij} = 1 \quad i = 1, 2, \ldots, n \\
& x_{ij} = \begin{cases} 1 & \text{if job } J_i \text{ is assigned to machine } M_j \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{1}$$

The first restriction assures $C_{\max}$ is the makespan of the schedule while the second one assures each job is executed only on one machine. According to the three-field notation in the scheduling fields (Graham *et al.* 1979), the IPMS problem is denoted as $P\|C_{\max}$, where P means the identical parallel machines and $C_{\max}$ designates the maximum completion time or makespan.

Considering the pre-emptive schedules, a lower bound (*LB*) value can be computed according to McNaughton (1959) when all machines finish their work at one moment for each IPMS problem, which is used as a criterion to evaluate the performance of the heuristics in this article. The definition of *LB* is given as follows:

$$LB = \left\{\sum_{i=1}^{n} p_i/m,\ \max_{1\le i\le n}(p_i)\right\} \tag{2}$$

3. Dynamic harmony search algorithm

3.1. *Basic harmony search algorithm*

The HS is a population-based optimization algorithm developed for solving optimization problems by mimicking the improvisation process of music players (Geem *et al.* 2001). In a HS system, there is a harmony memory (HM) that consists of a population of harmonies. Each harmony is regarded as a solution vector. Firstly, all harmonies in HM are initialized randomly. Secondly, a new harmony is improvised by using the following rules: memory consideration, pitch adjustment and random selection. Finally, the HM will be updated if the fitness value of a new harmony is better than that of the worst one in HM. The improvisation and updating process are repeated until a predefined stopping criterion is reached. The procedure of the algorithm is given as follows:

Step 1: Initialize algorithm parameters.
The HM parameters are initialized in this step, including harmony memory size (*HMS*), harmony memory considering rate (*HMCR*), pitch adjusting rate (*PAR*) and the number of improvisations (*NI*). Here, *HMS* is the number of solutions in the harmony memory. *HMCR* and *PAR* are parameters used to search for globally and locally improved solutions, respectively (Lee and Geem 2005).

Step 2: Initialize the harmony memory.
The initial HM matrix, shown in Equation (3), is filled with *HMS* number of harmonies. Each harmony, represented as an n-dimensional real-valued vector, is randomly generated by using a uniform distribution.

$$HM = \begin{bmatrix} X^1 \\ X^2 \\ \vdots \\ X^{HMS} \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_n^1 \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \vdots & \\ x_1^{HMS} & x_2^{HMS} & \dots & x_n^{HMS} \end{bmatrix} \tag{3}$$

Step 3: Improvise a new harmony.
Let $X' = \{x'_1, x'_2, \dots, x'_n\}$ represent a new harmony, which is generated in this step as improvization by using memory consideration, pitch adjustment and random selection.

In the memory consideration, each decision variable $x'_j (j \in [1, n])$ is chosen randomly from the jth column of the HM matrix with the probability of *HMCR* so that the historical information obtained during the past can be well used. With the probability of $1 - HMCR$, x'_j is generated randomly in the possible range as follows:

$$x'_j = \begin{cases} x'_j \in \{x_j^1, x_j^2, \dots x_j^{HMS}\} & \textit{with probability HMCR} \\ x_{j,lower} + rand() \times (x_{j,upper} - x_{j,lower}) & \textit{with probability } (1 - HMCR) \end{cases} \tag{4}$$

where $x_{j,upper}$ and $x_{j,lower}$ are the upper and lower bounds for each decision variable, respectively, and *rand*() is a random number uniformly generated between 0 and 1.

Furthermore, every component obtained by the memory consideration is adjusted by the pitch adjustment rule with the proportion of *PAR*. In the pitch adjustment, x'_j is changed as follows:

$$x'_j = x'_j \pm rand() \times BW \tag{5}$$

where BW is an arbitrary distance bandwidth.

Step 4: Update the harmony memory.
If the fitness value of the new harmony is better than that of the worst one in the HM, the HM will be updated. That is, the new generated harmony will replace the worst one to be a new member of the HM.

Step 5: Check the stopping criterion.
If the termination criterion is satisfied, terminate the algorithm and output the best solution; otherwise go back to Step 3.

3.2. *Dynamic harmony search algorithm*

Although the HS algorithm has been successfully applied to a wide variety of optimization problems, it is not efficient in performing local search for numerical applications (Mahdavi *et al.* 2007). Much research work has been done to improve the performance of the HS algorithm (Mahdavi *et al.* 2007, Cheng *et al.* 2008, Omran and Mahdavi 2008, Pan *et al.* 2010). Different from the existing version of HS, an improved HS based on dynamic sub-HMs and a new improvization process are proposed in this article.

3.2.1. *Dynamic sub-HMs*

It was pointed out that dynamic subpopulations can not only avoid the fast convergence, but also can increase the diversity of the population (Liang and Suganthan 2005). Therefore, the dynamic sub-HMs can be applied to improve the HS algorithm. That is, the whole HM is firstly divided into several sub-HMs, where the size of each sub-HM is set as a small value since a small-sized HM usually works better than a large one for the HS algorithm (Omran and Mahdavi 2008). Secondly, each sub-HM performs evolution independently in the solution space with its own members. However, the algorithm has a tendency to lose diversity and to be trapped into local optima due to its small size. Thus, a regrouping schedule is used to address this problem. That is, the harmonies from all sub-HMs are gathered together every R iterations and then regrouped randomly into different sub-HMs. Each sub-HM restarts searching by using the new configuration of small HM. In this way, information found by all the sub-HMs in previous stage can be exchanged periodically among the harmonies. Accordingly, the diversity of each sub-HM can be increased, and the premature convergence can be avoided also. Figure 1 shows how the dynamic HS algorithm works.

3.2.2. *Improved harmony improvisation*

In this section, an improved improvisation scheme is presented. Different from the original HS, where each harmony has the same probability to be chosen in the improvisation process, the proposed method is based on such an idea that the better the performance of a harmony, the

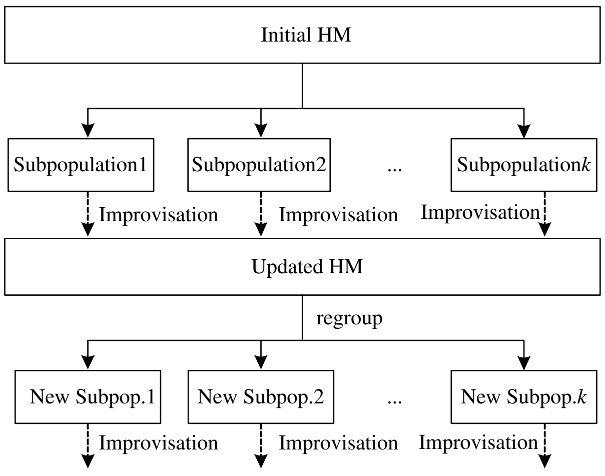

Figure 1. HS search with dynamic sub-HM topology.

greater is the probability that it will be selected (Cheng *et al.* 2008). So, the local-best harmony X^{lBest} is used to produce a new harmony X' in each sub-HM and the memory consideration is correspondingly updated as follows:

$$x'_j = x_j^{lBest} \quad j = 1, 2, \ldots, n \tag{6}$$

In this way, the population may converge to a local optimum quickly. To increase the diversity and slow down the convergence rate, the pitch adjustment rule is modified as follows.

$$x'_j = x_j^{rws} \pm rand() \times BW \quad rws \in (1, 2, \ldots, SHMS) \tag{7}$$

where x_j^{rws} is the jth decision variable of harmony X^{rws} selected by a roulette wheel selection in each sub-HM and *SHMS* is the sub-HM size. The selection probability of harmony X^i is computed as follows:

$$p(X^i) = (F_{max}(X) - F(X^i) + 1) / \sum_{i=1}^{SHMS} (F_{max}(X) - F(X^i) + 1) \tag{8}$$

where $F(X^i)$ denotes the objective value of X^i, $F_{max}(X)$ is the maximum objective value obtained in each sub-HM. It is obvious that a harmony with smaller value has a larger chance than the others to be selected. During the roulette wheel selection, a uniformly random number r is generated in the range [0, 1]. Then the harmony X^{rws} will be chosen if the following criterion is satisfied:

$$\sum_{i=1}^{rws-1} p(X^i) < r \leq \sum_{i=1}^{rws} p(X^i) \tag{9}$$

The flow chart of the improved improvisation scheme is shown in Figure 2.

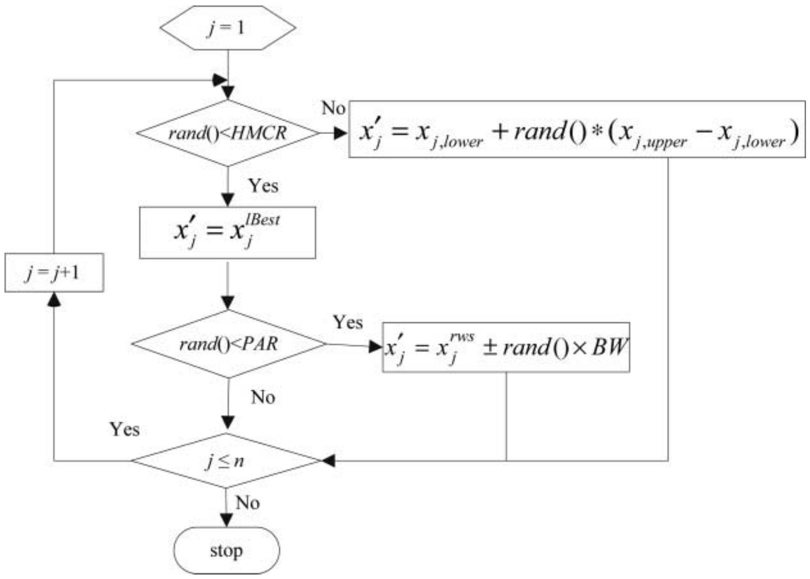
$j=1$
rand()<HMCR
No
$x'_j = x_{j,lower} + rand() * (x_{j,upper} - x_{j,lower})$
Yes
$x'_j = x_j^{lBest}$
$j=j+1$
Yes
rand()<PAR
$x'_j = x_j^{rws} \pm rand() \times BW$
No
Yes
$j \leq n$
No
stop

Figure 2. The improved improvisation scheme of the dynamic HS algorithm.

4. Hybrid dynamic harmony search algorithm for IPMS problems

4.1. *Solution representation*

Due to the continuous nature of the HS, it cannot be directly applied to the scheduling problems considered in this article. Therefore, a suitable mapping scheme should be designed to construct a direct relationship between discrete domain space and the real-valued harmonies. In this article, n numbers of harmony dimensions are designed for n numbers of jobs. Each dimension represents a typical job. The continuous value of each harmony dimension will be converted to a discrete machine number to which the corresponding job is assigned by applying a LS-based rule. In particular, the rank of each decision variable is used to represent a scheduling order so that all jobs can be allocated to the least loaded machine one by one. That is, the job with the largest decision variable is firstly allocated to a machine with minimal workload (If there is more than one such machine, select one randomly). Then the job with the second largest decision variable will be allocated to a machine with currently minimal workload. This will be done repeatedly until all remaining jobs are assigned. In this way, an allocation scheme is constructed, and the harmony can be evaluated subsequently.

An example is shown in Table 1 for a scheduling problem with seven jobs and three machines. Suppose a harmony is $X = [0.45, 0.11, 0.78, 0, 92, 0.6, 0.03, 0.67]$. Since $x_4 = 0.92$ is the largest decision variable, job 4 is firstly assigned to machine 1. Then, job 3 with $x_3 = 0.78$ is assigned to machine 2. In the same way, the remaining jobs 7, 5, 1, 2 and 6 are assigned to machines 3, 2, 1, 3 and 1, respectively. Thus, a solution $S = [1, 3, 2, 1, 2, 1, 3]$ is obtained.

4.2. *Initial HM*

To generate the initial HM with a certain level of quality and diversity, the LPT rule is used to generate a harmony vector while the other *HMS*-1 harmonies are generated uniformly in the search space. Since the solution obtained by the LPT is represented by discrete values, it should be converted to continuous decision variables for further evolution. The conversion is given as follows:

$$x_i = x_{i,lower} + (x_{i,upper} - x_{i,lower}) * p_i/p_{max} \quad i = 1, 2, \ldots, n \tag{10}$$

where x_i is the ith decision variable of harmony X, p_i is the processing time of job J_i, p_{max} denotes the largest processing time among the jobs, $x_{i,lower}$ and $x_{i,upper}$ are set as 0 and 1, respectively.

Clearly, such a harmony and its corresponding schedule obey the LS rule because the jobs are sorted by the LPT heuristic according to their processing times. After the whole HM is generated, it is divided into multiple sub-HMs with the same size *SHMS* randomly.

4.3. *VNS-based local search*

VNS is an effective meta-heuristic for solving optimization problems, whose basic idea is systematic change of neighbourhood combined with a local search (Mladenovic and Hansen

Table 1. Illustration of decision variables of a harmony and the corresponding solution obtained by the list scheduling rule.

Job	1	2	3	4	5	6	7
Processing time	2	3	2	3	6	4	3
Decision variable	0.45	0.11	0.78	0.92	0.6	0.03	0.67
Solution	1	3	2	1	2	1	3

1997). In this article, a VNS-based local search is designed to improve the effectiveness of the HS algorithm.

The neighbourhood structure is one of the key elements of VNS. For a given harmony (schedule), all the machines can be classified into two categories: problem machines and non-problem machines. The problem machines are the machines with their finish time equal to the makespan of the solution, while the remaining machines are regarded as non-problem machines. For the IPMS problem, it is clear that the quality of the given solution can be improved by moving or swapping jobs between the problem machines and any other non-problem machines. Here, four different kinds of neighbourhoods are sequentially utilized, *i.e*. *Move* (move one job from a problem machine to a non-problem machine), *Swap* (swap one job from a problem machine with one job from a non-problem machine), *Asymmetric Swap* (swap one job from a problem machine with two jobs from a non-problem machine), and *Double Swap*(swap two jobs from a problem machine with two jobs from a non-problem machine). A simple example is provided in Figure 3 to illustrate the above four neighbourhood structures. For the aforementioned problem with 7 jobs and 3 machines, a given solution is $S = [1, 3, 2, 1, 2, 1, 3]$ with a makespan of 9. From Figure 3 it can be found that machine M_1 is a problem machine, while machine M_2 and M_3 are two non-problem machines. The following local searches are all performed between the problem machine and non-problem machines. For the *Move* operator, a job J_1 is selected from problem machine M_1 and moved to non-problem machine M_3 (Figure 3(a)). For the *Swap* operator, a job J_6 is selected from problem machine M_1 and swapped with J_7 from non-problem machine M_3 (Figure 3(b)). For the *Asymmetric Swap* operator, two jobs, J_1 and J_4, from problem machine M_1 are swapped with J_7 from non-problem machine M_3 (Figure 3(c)). For the *Double Swap* operator, two jobs, J_4 and J_6, from problem machine M_1 are swapped with other two jobs, J_2 and J_7, from non-problem

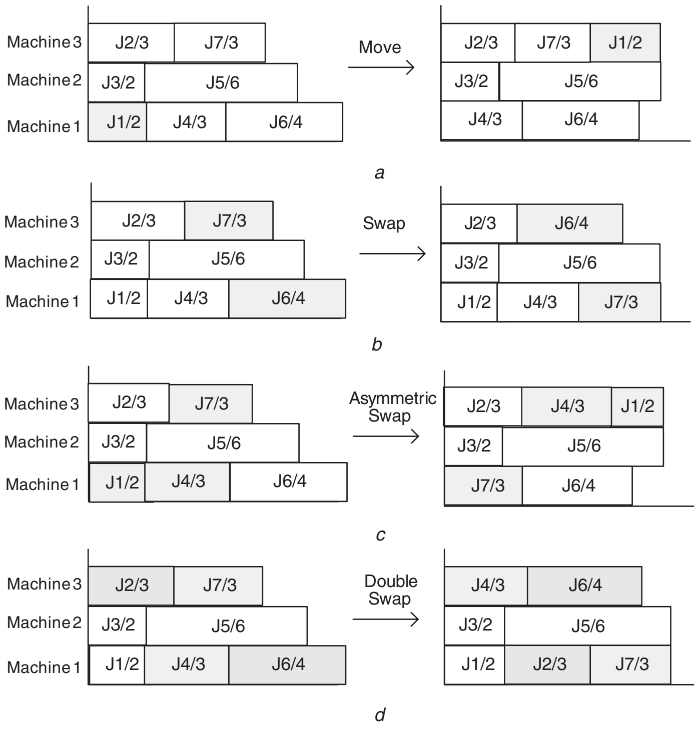

Figure 3. Four neighbourhood structures for local search.

machine M_3 (Figure 3(d)). It is obvious that the makespan of this problem is reduced from 9 to 8 and a better solution can be obtained after performing one of the four local searches.

The local search methods in the present algorithm are not directly applied to decision variables but to machines to which jobs are assigned. So, when performing these local searches on the selected jobs, decision variables should be adjusted in correspondence with the change of the schedule. Fortunately, during the local search, the adjustment is very simple if a given job is swapped with another job. That is to say, when the two selected jobs are swapped with each other, the solution should be adjusted by swapping the machines to which they are assigned. Accordingly, the decision variables of the two jobs are swapped at the same time. When a job J_i is moved to another machine by using the *Move* or the *Asymmetric Swap* operator, the corresponding decision variable of J_i is just kept at its original value.

The presented VNS-based local search is illustrated as follows:

```
Step 1: Initialization: set initial schedule $S$, $MaxIterNum = 100, loop = 1, k_{max} = 4$.
Step 2: Compute $LB$ according to Equation (2).
Step 3: Repeat until $C_{\max}(S)$ is equal to $LB$ or $loop > MaxIterNum$.
    Step 3.1: For the schedule $S$, sort the machines in descending order according to their
              finish time. Determine the problem machine set ($MS_p$) and the non-problem
              machine set ($MS_{np}$).
    Step 3.2: For each machine $Ma$ in $MS_p$ do.
        Step 3.2.1: For each machine $Mb$ in $MS_{np}$ do
            Step 3.2.1.1: Set $k = 1$;
            Step 3.2.1.2: Repeat until ($k > k_{\max}$)
                Step 3.2.1.2.1: if ($k = 1$) then
                                $S' = Move(S, Ma, Mb)$;
                Step 3.2.1.2.2: if ($k = 2$) then
                                $S' = Swap(S, Ma, Mb)$;
                Step 3.2.1.2.3: if ($k = 3$) then
                                $S' = ASwap(S, Ma, Mb)$;
                Step 3.2.1.2.4: if ($k = 4$) then
                                $S' = DSwap(S, Ma, Mb)$;
                Step 3.2.1.2.5: Evaluate $f(S')$; if $f(S') < f(S)$,
                                then set $S = S'$, go to step 3; oth-
                                erwise set $k = k + 1$.
    Step 3.3: Set $loop = loop + 1$
Step 4: Output the best solution found so far.
```

In order to reduce computational complexity of the method, the local search for each neighbourhood will be stopped immediately after an improvement on the given solution is obtained. For example, $S' = Swap\ (S, Ma, Mb)$ means analysing the swaps of different jobs belongs to problem machine *Ma* and non-problem machine *Mb* in the solution S. The searching process will be stopped once a better solution S' is found or no improvement can be made. The pseudo-code of the *Swap* function is given as follows:

```
Step 1: flag = false;
Step 2: For (each $J_a \in Ma$ and $flag$ = false)
            For (each $J_b \in Mb$ and $flag$ = false)
                If solution considering $J_a$ swap with $J_b$ is better than current solution $S$
                swap $J_a$ with $J_b$, $flag$ = true.
```

In the algorithm, every harmony is selected as an initial solution to be improved by the VNS-based local search algorithm with the probability of 0.1. After the local search is completed, the corresponding sub-HM will be updated if a solution with better quality is found. Combined with the local search strategy, the dynamic harmony search (DHS) algorithm is modified to DHS with local search (DHS_LS).

4.4. *DHS_LS algorithm for IPMS problem*

Based on the proposed dynamic sub-HM concept, new harmony improvisation scheme and VNS-based local search strategy, the framework of DHS_LS algorithm for IPMS problem is provided as follows:

```
Step 1: Set HMCR, PAR, BW, NI, R, set iternum = 0.
Step 2: Initialize HMS and HM. For each harmony, determine the corresponding allocation
        scheme by LS-based rule.
Step 3: Divide the HM into multiple sub-HMs with the same size SHMS randomly.
Step 4: Repeat until iternum is equal to NI:
        Step 4.1: For each sub-HM perform the following steps.
                  Step 4.1.1: Improvise a new harmony $X^{new}$ as described in Section 3.2.
                  Step 4.1.2: For the new harmony, determine the corresponding allocation
                              scheme S by LS-based rule and compute its makespan.
                  Step 4.1.3: Perform VNS-based local search described in Section 4.3 on $X^{new}$
                              with the probability 0.1.
                  Step 4.1.4: Update the sub-HM if $X^{new}$ is better than the worst harmony in the
                              sub-HM.
        Step 4.2: If the regrouping period R is reached, regroup the HM randomly.
        Step 4.3: Set iternum = iternum + 1.
Step 5: Output the best harmony found so far.
```

5. Simulation results and comparisons

5.1. *Experimental setup*

In this section, four experimental frameworks are examined taken from Gupta *et al.* (2001), Lee *et al.* (2006) and Kashan and Karimi (2009) to test the performance of the proposed algorithms. Table 2 presents a summary of four experimental frameworks, where each experimental framework includes three variables: the number of machines (m), the number of jobs (n), and the minimum and maximum values to uniformly generate processing times. As a result, 120 experimental conditions are conducted and each condition is replicated 50 times with different data. This yields a total number of 6000 problems.

Table 2. Summary of the computational experiments.

	m	n	p
E1	3, 4, 5	2m, 3m, 5m	U(1, 20), U(20, 50)
E2	2, 3, 4, 6, 8, 10	10, 30, 50, 100	U(100, 800)
E3	3, 5, 8, 10	3m + 1, 3m + 2, 4m + 1, 4m + 2, 5m + 1, 5m + 2	U(1, 100), U(100, 200), U(100, 800)
E4	2 3	9 10	U(1, 20), U(20, 50), U(50, 100), U(100, 200), U(100, 800)

The DHS_LS algorithm was coded in Visual C++ 6.0 and run on a Pentium 2 CPU 2.0 GHz with 2 GB of RAM. For the DHS_LS, the parameters are set as $R = 5$, $SHMS = 3$. The number of sub-HMs varies from 3 to 10 for different problems. The other parameters are set as: $HMCR = 0.9$, $PAR = 0.3$, $BW = 0.01$ and $NI = 100$. The algorithm is stopped either after running for NI iterations or when getting the LB value.

The mean performance (MP) and the average execution time (AET) of the algorithm are used for comparison, which are computed as follows:

$$MP = \sum_{i=1}^{n} (C_{max})_i / \sum_{i=1}^{n} LB_i \tag{11}$$

$$AET = \left(\sum_{i=1}^{n} Et_i \right) / n \tag{12}$$

where $n = 50$ is the number of instances for each parameter combination, and $(C_{max})_i$, LB_i and Et_i are the makespan, the lower bound and the execution time of the ith out of 50 instances, respectively.

Besides, the relative performances of the DHS algorithm and the DHS_LS algorithm with respect to other heuristics are presented. For example, a value of c/d in column DPSO/DHS means that out of all the 50 problems, DPSO yields better solutions than DHS for c problems, DHS performs better for d problems, and DPSO and DHS yield the same solutions 50–c–d problems.

5.2. *Comparisons of BHS, DHS and DHS_LS*

First, the effectiveness is demonstrated of dynamic sub-HMs and local search by comparing DHS_LS with DHS and BHS (basic harmony search). For the BHS algorithm, the default parameters are set the same as those in Omran and Mahdavi (2008). The statistical results for experiment E2 are listed in Table 3.

From Table 3, it can be seen that the mean performance obtained by DHS is much better than that by BHS. DHS obtains better solutions for all instances, which shows the effectiveness of introducing dynamic sub-HMs into DHS. In addition, DHS_LS finds better solutions than DHS for 877 out of 1000 tested problems, and reports no inferior solutions in these instances. So, the performance of DHS has been further improved by incorporating the local search. In terms of average execution time, it may find that there are some instances in which the time spent by DHS_LS is less than DHS. This is due to the fact that the extensive use of local search speeds up the convergence rate of DHS_LS.

5.3. *Comparisons of DHS_LS, SA and HDPSO*

Next, DHS_LS is compared with three existing algorithms, *i.e.* SA (Lee *et al.* 2006), DPSO and hybrid version of DPSO (HDPSO) (Kashan and Karimi 2009). The SA algorithm (Lee *et al.* 2006) is briefly reviewed as follows. First, the algorithm starts evolution from an initial solution with the LPT sequence. Then it generates a new solution by swapping the positions of two selected jobs. The new solution is accepted if its makespan is better than that of the current solution; otherwise, it is accepted with some probability which decreases as the process evolves. This process is repeated until the stopping criterion is reached. Computational results showed that the SA method worked better than the LISTFIT (Gupta *et al.* 2001) and the PI (Fatemi *et al.* 1998) heuristics. In the DSPO (Kashan and Karimi 2009) method, both the particle representation and the equations of

Table 3. Mean performance (MP) and average execution time (AET) generated by BHS, DHS and DHS_LS.

		BHS		DHS		DHS_LS				
m	*n*	MP	AET	MP	AET	MP	AET	BHS/DHS	BHS/ DHS_LS	DHS/ DHS_LS
2	10	1.0172	0.021	1.0010	0.014	1.0009	0.354	0/50	0/50	0/26
3		1.0610	0.023	1.0139	0.017	1.0116	0.286	0/50	0/50	0/38
2	30	1.0056	0.055	1.0000	0.024	1.0000	0.007	0/50	0/50	0/42
3		1.0185	0.060	1.0006	0.142	1.0000	0.030	0/50	0/50	0/50
4		1.0350	0.064	1.0025	0.154	1.0000	0.095	0/50	0/50	0/50
6		1.0782	0.072	1.0097	0.167	1.0000	0.069	0/50	0/50	0/50
8		1.1297	0.079	1.0194	0.181	1.0005	0.062	0/50	0/50	0/50
10		1.1741	0.034	1.0327	0.076	1.0034	0.050	0/50	0/50	0/50
2	50	1.0033	0.091	1.0000	0.052	1.0000	0.021	0/50	0/50	0/7
3		1.0111	0.098	1.0004	0.236	1.0000	0.022	0/50	0/50	0/50
4		1.0216	0.104	1.0015	0.250	1.0000	0.027	0/50	0/50	0/50
6		1.0468	0.116	1.0057	0.273	1.0000	0.090	0/50	0/50	0/50
8		1.0763	0.078	1.0117	0.180	1.0000	0.196	0/50	0/50	0/50
10		1.1030	0.054	1.0198	0.122	1.0000	0.723	0/50	0/50	0/50
2	100	1.0017	0.177	1.0000	0.110	1.0000	0.035	0/50	0/50	0/14
3		1.0059	0.193	1.0002	0.457	1.0000	0.040	0/50	0/50	0/50
4		1.0109	0.206	1.0008	0.492	1.0000	0.042	0/50	0/50	0/50
6		1.0236	0.229	1.0029	0.533	1.0000	0.047	0/50	0/50	0/50
8		1.0362	0.105	1.0061	0.242	1.0000	0.240	0/50	0/50	0/50
10		1.0525	0.107	1.0099	0.244	1.0000	0.029	0/50	0/50	0/50
Average		1.0456	0.098	1.0069	0.198	1.0008	0.123	0/50	0/50	0/43.9

generating a new particle are reconstructed in a discrete domain space. A particle's position is represented by an array whose elements are discrete machine numbers. Although the velocity and the position equations have analogous structures to those of the original PSO, the operators have been assigned different meanings to be adaptive to discrete-valued operands. The DPSO is also hybridized with a local search heuristic. The hybrid version is denoted as HDPSO, which showed competitive performance when compared with the SA method (Lee *et al.* 2006) through a large amount of experiments. All the compared algorithms in this work are coded in Visual C++ according to the pseudo-code shown in their original article, and the parameters are set to the values recommended by the authors.

Table 4 reports the overall mean performance (*OMP*) and the overall average execution time (*OAET*) for each experimental framework listed in Section 5.1. For experiment E1, it can be found that the results obtained by DHS_LS are almost the same as HDPSO in terms of the mean performance. Comparing DHS with DPSO (two heuristics without hybridizing local search elements), DHS performs better 17.8% (8.9/50) and worse 0.2% (0.1/50) of the time. When DHS_LS is compared to SA, it is obvious that DHS_LS yields better solutions than SA for 29.2% (14.5/50) of the time while both DLS_LS and SA find the same solutions for the remaining instances. When DHS is compared to SA, there are some experiments which DHS performs better than SA and vice versa. In terms of the average execution time, both DHS and DHS_LS take less time than SA, DPSO and HDPSO.

For experiment E2, it can be easily observed that the DHS_LS algorithm outperforms the other heuristics. That is to say, the DHS_LS performs better than HDPSO for 81.8% (40.9/50) of the time, and provides better performance than SA for almost all the instances. In addition, it is noticed that the mean performance of DHS is superior to those of DPSO and SA in most cases. As the number of jobs and the ranges of the processing times get larger, the problems become more difficult. However, the DHS_LS reaches optimal solutions with faster convergence rate than SA and HDPSO as usual.

Table 4. Overall mean performance (OMP) and overall average execution time (OAET) generated by the compared algorithms for experiments E1-E4.

	SA		DPSO		HDPSO		DHS		DHS_LS					
	OMP	OAET	OMP	OAET	OMP	OAET	OMP	OAET	OMP	OAET	SA/DHS	SA/DHS_LS	DPSO/DHS	HDPSO/ DHS_LS
E1	1.0286	2.141	1.0239	1.276	1.0204	1.189	1.0215	0.011	1.0204	0.036	6.8/13.3	0/14.6	0.1/8.9	0/0.8
E2	1.0136	3.595	1.0435	2.883	1.0011	2.315	1.0081	0.094	1.0008	0.155	4.2/39.6	0/49.3	1.4/47.4	0.1/40.9
E3-1	1.0269	2.220	1.0709	1.957	1.0017	1.599	1.0148	0.054	1.0011	0.044	11.5/32.6	0/47.7	1.3/47	0/25.7
E3-2	1.0834	3.057	1.0874	2.183	1.0024	1.743	1.0174	0.058	1.0022	0.069	0.6/48.1	0/50	0.2/49.8	0/28
E3-3	1.0342	2.419	1.0798	2.010	1.0023	1.642	1.0169	0.054	1.0014	0.455	5/45	0/50	1.1/48.8	0.2/46
E4	1.0460	1.963	1.0080	1.278	1.0061	1.173	1.0064	0.006	1.0061	0.077	0.8/42	0/42.8	1.9/14.4	0/2.8

In Table 4, E3-1, E3-2, and E3-3 represent the results of the experiment E3 when the processing times are generated by $U(1,100)$, $U(100,200)$, and $U(100,800)$, respectively. These problems are classified as difficult problems by Gupta *et al.* (2001). It can be concluded that the superiority of DHS_LS over DPSO and SA is more significant than in the above experiments. That is, DHS_LS performs better 98.5% (147.7/150) of the time compared to SA and 66.5% (99.7/150) to HDPSO. This reveals the fact that the performance of the DHS_LS is very stable when the problems get more difficult. On the other hand, DHS performs very well in this experiment compared to SA and DPSO. Furthermore, the execution time spent by all heuristics increases as the scale of the problems becomes larger. It is seen that the DHS takes less time than other heuristics in most of the instances. However, when the processing times are generated from $U(1,100)$, DHS_LS takes less time than DHS because of its better local search ability.

Since the problems in experiment E4 are very simple, it can be seen from Table 4 that both HDPSO and DHS_LS reveal good performance, and DHS_LS performs slightly better than HDPSO. Results also demonstrate that both DHS and DHS_LS outperform SA in most of the experiments. The average execution time taken by DHS_LS and DHS is less than those of the SA and the PSO algorithms as before.

5.4. *Parameter analysis*

Finally, the effect of *SHMS* (the number of harmonies included in each sub-HM), *SHMnum* (the number of sub-HMs) and *R* (regrouping frequency) on the performance of the DHS algorithm is investigated by using experiment E2. In the experiment, *SHMS* is set as 3, 5, 10, 30 or 50; *SHMnum* is set as 3, 5, 10, 20 or 50; *R* varies from 5 to 25 with a step equal to 5. Figure 4 summarizes the results obtained using different values for these parameters. From Figure 4, it can be seen that, as *SHMS* increases the mean performance increases. That is, the performance of DHS depends on sub-HM size, but it is not too much great. DHS reaches its best performance when $SHMS = 3$ (T1), which is recommended in the presented DHS.

The parameter *SHMnum* decides the number of sub-HMs included in the HM. Results show that a large *SHMnum* value can increase the diversity of the HM and help the DHS algorithm to find better solutions, but it also slows down the convergence rate and increases the computational time, while a small *SHMnum* value cannot provide sufficient and diverse information for the

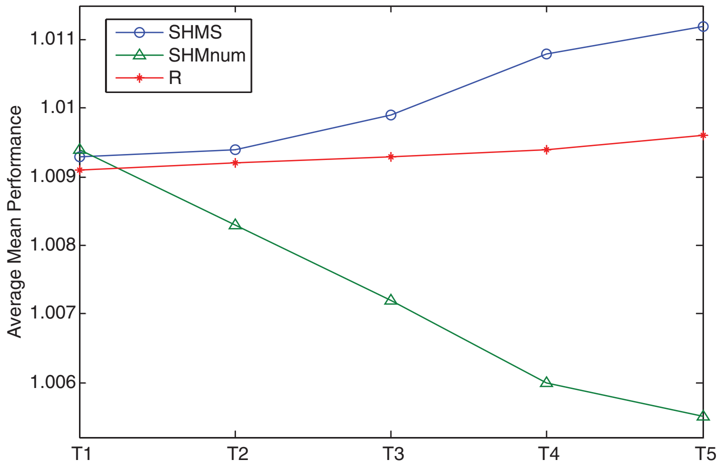

Figure 4. The effect of parameters SHMS (sub-HM size), SHMnum (number of sub-HMs) and R (regrouping frenquency) on the performance of DHS.

generation of the new harmony, the mean performance of the DHS is not as good as that obtained by a larger *SHMnum* value. So, $SHMnum = 3$ (T1) is recommended for problems with small size while a larger value is suggested for problems with medium or large size.

From Figure 4, it is observed that as R varies from 5 to 25 the DHS algorithm generates very similar results, which suggests that DHS is less sensitive to the parameter R. It can be seen that the mean performance obtained by DHS when $R = 5$ is a bit better than those obtained when $R > 5$. So, $R = 5$ (T1) is recommended.

6. Conclusions

A novel dynamic harmony search (DHS) algorithm was proposed in this article for solving identical parallel machines scheduling problem to minimize makespan. To the best of the authors' knowledge, this was the first report to apply the HS algorithm to the IPMS problem. With the characteristics of dynamic multi sub-HMs, regrouping scheme and new improvisation method, the proposed DHS algorithm was of good performance on a large scale of problems. Moreover, DHS was hybridized with a VNS-based local search (DHS_LS) to enhance the exploitation ability. The experimental results showed that the DHS_LS algorithm outperformed the SA and the HDPSO heuristics, whereas it could find optimal solutions almost for all tested problems. In addition, DHS_LS performs with an excellent computation speed. In summary, the proposed harmony search produces competitive performances in effectiveness and efficiency for solving IPMS problems. The future work is to apply the algorithm to other kinds of scheduling problems, such as the unrelated parallel machines problems and the precedence related job scheduling problems.

Acknowledgements

This research is partly supported by National Science Foundation of China under grants 60874075, 70871065 and 60834004, and Science Foundation of Shandong Province, China (BS2010DX005), and Doctoral Program Foundation of Institutions of Higher Education of China (20100002110014), and Program for New Century Excellent Talents in University (NCET-10-0505) and Project of Shandong Province Higher Educational Science and Technology Program under grants J09LG29 and J10LG25.

References

Cheng, Y.M., Li, L., Lansivaara, T., Chi, S.C. and Sun, Y.J., 2008. An improved harmony search minimization algorithm using different slip surface generation methods for slope stability analysis. *Engineering Optimization*, 40, 95–115.

Coffman, E.G., Garey, M.R., and Johnson, D.S., 1978. An application of bin-packing to multiprocessor scheduling. *SIAM Journal of Computing*, 7, 1–17.

Costa, A.M., Vargas, P.A., Von Zuben, F.J., and Franca, P.M., 2002. Makespan minimization on parallel processors: an immune-based approach. *Proceedings of the 2002 congress on evolutionary computation (CEC2002)*, 12–17 May, Honolulu, HI, USA. New York: IEEE Press, 920–925.

Ghomi, S.M.T. and Ghazvini, F., 1998. A pairwise interchange algorithm for parallel machine scheduling. *Production Planning & Control*, 9, 685–689.

Garey, M.R. and Johnson, D.S., 1979. *Computers and intractability: a guide to the theory of np completeness*. San Francisco, CA: Freeman.

Geem, Z.W., Kim, J.H. and Loganathan, G.V., 2001. A new heuristic optimization algorithm: harmony search. *Simulation*, 76, 60–68.

Geem, Z.W., Lee, K.S. and Park, Y.J., 2005. Application of harmony search to vehicle routing. *American Journal of Applied Sciences*, 2, 1552–1557.

Graham R.L., 1969. Bounds on multiprocessor timing anomalies. *SIAM Journal of Applied Mathematics*, 17, 416–429.

Graham, R.L.E., Lawler, L., Lenstra, J.K. and Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *ADM*, 5, 287–326.

Gupta, J.N.D. and Ruiz-Torres, J., 2001. A LISTFIT heuristic for minimizing makespan on identical parallel machines. *Production Planning & Control*, 12, 28–36.

Kashan, A.H. and Karimi, B., 2009. A discrete particle swarm optimization algorithm for scheduling parallel machines. *Computers & Industrial Engineering*, 56, 216–223.

Lee, C.Y. and Massey, J.D., 1988. Multiprocessor scheduling: combining LPT and MULTIFIT. *Discrete Applied Mathematics*, 20, 233–242.

Lee, K.S. and Geem, Z.W., 2005. A new meta-heuristic algorithm for continuous engineering optimization, harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 194, 3902–3933.

Lee, W.C., Wu, C.C. and Chen, P., 2006. A simulated annealing approach to makespan minimization on identical parallel machines. *International Journal of advanced Manufacturing Technology*, 31, 328–334.

Liang, J.J. and Suganthan, P.N., 2005. Dynamic multi-swarm particle swarm optimizer. *Proceedings of IEEE international swarm intelligence symposium (ISIS 2005)*, 8–10 June, Pasadena, CA, USA. New York: IEEE Press, 124–129.

Liu, M., and Wu, C., 1999. A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering*, 13, 399–403.

Mahdavi, M., Fesanghary, M. and Damangir, E., 2007. An improved harmony search algorithm for solving optimization problems. *Applied Mathematics and Computation*, 188, 1567–1579.

McNaughton, R., 1959. Scheduling with deadlines and loss functions. *Management Science*, 6, 1–12.

Mladenovic, N. and Hansen, P., 1997. Variable neighborhood search. *Computers and Operations Research*, 24 (11), 1097–1100.

Omran, M.G.H. and Mahdavi, M., 2008. Global-best harmony search. *Applied Mathematics and Computation*, 198, 643–656.

Pan, Q.K., Suganthan, P.N., Fatih Tasgetiren, M. and Liang, J.J., 2010. A self-adaptive global best harmony search algorithm for continuous optimization problems. *Applied Mathematics and Computation*, 216 (2), 830–848.

Sevkli, M. and Uysal, H.A., 2009. A modified variable neighborhood search for minimizing the makespan on identical parallel machines. *Proceedings of the international conference on computers & industrial engineering (ICCIE 2005)*, 6–9 July, Troyes, France. New York: IEEE Press, 108–111.