Ann Oper Res (2012) 194:3–31
DOI 10.1007/s10479-010-0769-z

A harmony search algorithm for university course timetabling

Mohammed Azmi Al-Betar · Ahamad Tajudin Khader

Published online: 10 July 2010

Abstract One of the main challenges for university administration is building a timetable for course sessions. This is not just about building a timetable that works, but building one that is as good as possible. In general, course timetabling is the process of assigning given courses to given rooms and timeslots under specific constraints. Harmony search algorithm is a new metaheuristic population-based algorithm, mimicking the musical improvisation process where a group of musicians play the pitches of their musical instruments together seeking a pleasing harmony. The major thrust of this algorithm lies in its ability to integrate the key components of population-based methods and local search-based methods in a simple optimization model. In this paper, a harmony search and a modified harmony search algorithm are applied to university course timetabling against standard benchmarks. The results show that the proposed methods are capable of providing viable solutions in comparison to previous works.

Keywords Course timetabling · Harmony search · Metaheuristic algorithms · Exploration · Exploitation

1 Introduction

University timetabling is a demanding and challenging repetitive administrative task for academic institutions. In general, timetabling is the process of allocating given events, each with given features, to given resources and times with respect to given constraints (Burke et al. 2004). The timetabling process varies in difficulty according to the problem size and demanding constraints which vary among academic institutions. The timetabling solution is typically evaluated against satisfying constraints. Constraints are usually categorized into two types (Burke et al. 1997): hard and soft. Hard constraints must essentially be satisfied in

M.A. Al-Betar (✉) · A.T. Khader
School of Computer Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia
e-mail: mohbetar@cs.usm.my

A.T. Khader
e-mail: tajudin@cs.usm.my

the timetabling solution to be feasible, whereas soft constraints are desired but not absolutely essential. Soft constraints may be violated. Yet the more they are met in the timetabling solution, the better the quality of the solution. University timetabling is usually divided into two problems: the exam timetabling problem and the course timetabling problem which is the concern of this paper.

The course timetabling problem has been given particular attention by operational research and artificial intelligence experts for quite a long time. Many methods have been introduced in the literature to tackle such a problem. As is commonly known, the basic timetabling problem can be modeled as a graph coloring problem (i.e., an undirected graph involves given vertices, each of which reflects one event; the colour of each vertex reflects a particular timeslot, and an edge between vertices reflects the conflicting events which must be assigned different colours (or timeslots)). Therefore, the earliest methods employed graph coloring heuristics as an essential part to construct the course timetabling solution. These heuristics assign the courses to rooms and timeslots one by one according to a particular order. A backtracking algorithm is often used as a recovery approach for unscheduled events in the constructed solution (Carter et al. 1996). Although these heuristics show great efficiency in constructing a timetabling solution quickly, the quality of the solution is often inferior to that produced by metaheuristic or hyper-heuristic methods. Nowadays, these heuristics are normally used in the construction of initial solution(s) for metaheuristic methods; they are also employed in hyper-heuristic approaches as low level heuristics (Burke et al. 2003a, 2007). Asmuni et al. (2005) however used them collaboratively but were guided by a fuzzy assignment function to construct a '*good*' quality solution to the other methods.

The emergence of metaheuristics for solving difficult timetabling problems has been one of the most notable accomplishments of the last two decades or even earlier. Commonly, metaheuristics are divided into two categories, local search-based and population-based methods. The local search-based methods consider one solution at a time (Blum and Roli 2003). The solution undergoes changes iteratively until a final solution which is usually in the same region of the search space as the initial solution is reached. They often use neighborhood structures guided by a given acceptance rule to improve the solution. Although the main merit of using these methods is their strength of fine-tuning the solution more structurally and more quickly than population-based methods (Blum and Roli 2003), the main drawback is that they have a tendency to get stuck in a small region of the search space. This is mainly due to local search-based methods focusing on exploitation rather than exploration, which means that they move in one direction without performing a wider scan of the entire search space. The local search-based methods applied to the course timetabling problem include Iterative Local Search (Socha et al. 2002), Simulated Annealing (Chiarandini et al. 2006; Kostuch 2005), Very Large Neighborhood Search (Abdullah et al. 2007b, 2005), Great Deluge (McMullan 2007; Landa-Silva and Obit 2008, 2009; Obit et al. 2009; Turabieh et al. 2009).

Population-based metaheuristics have also been applied to the course timetabling problem. The population-based methods consider many solutions at a time. During the search, they recombine current solutions to obtain new ones. Unfortunately, the solutions produced by population-based methods are usually inferior to those produced by local search-based methods because they are poorer at finding the precise optimal solution in the search space region to which the algorithm converges (Fesanghary et al. 2008). The common reason for this problem is that the population-based methods are more concerned with exploration rather than exploitation. Recall that population based methods scan the solutions in the entire search space without rigorous concentration on current solutions in addition to other drawbacks such as the need for more time (Chiarandini et al. 2006).

Furthermore, according to building block theory (Goldberg 1989), the algorithm is assumed to be working well when the adjacent variable of the chromosomes are strongly correlated. In timetabling problems, however, this assumption does not seem plausible. This is why several timetable researchers have lately started focusing their attention on local search-based rather than the population-based methods (Abdullah et al. 2007b; Chiarandini et al. 2006). The population-based methods applied to the course timetabling problem include Genetic Algorithm (Lewis and Paechter 2004, 2005), Ant Colony Optimization (Socha et al. 2002), and Artificial Immune System (Malim et al. 2006). Overviews of previous methods for the course timetabling problem are available in the following surveys (Burke et al. 1997, 2004; Carter and Laporte 1997; Burke and Petrovic 2002; Lewis 2008).

In light of the above, a possible way to design an efficient algorithm that tackles the university course timetabling, is to strike a balance between global wide-ranging exploration using the strength of population-based methods, and local nearby exploitation using the strength of local search-based methods. Memetic Algorithms, which do attempt to combine the best features of both types of approach have also been applied for timetabling (Burke and Landa-Silva 2005).

With respect to the significant differences between the examination and course timetabling problems (McCollum 2006), in the recent comprehensive survey of examination timetabling, Qu et al. (2009) conclude: "There are many research directions generated by considering the hybridization of meta-heuristic methods particularly between population based methods and other approaches". In general, there are many research trends highlighting the efficiency of using local search-based methods within population-based methods. For example, Blum and Roli (2003) in an influential article on metaheuristics write "In summary, population-based methods are better in identifying promising areas in the search space, whereas trajectory methods are better in exploring promising areas in the search space. Thus, metaheuristic hybrids that in some way manage to combine the advantage of population-based methods with the strength of trajectory methods are often very successful". For course timetabling problem, a hybrid Evolutionary Algorithm with Variable Neighborhood Structure is developed by Abdullah et al. (2007a) with very successful outcomes.

The harmony search algorithm is a new metaheuristic algorithm developed by Geem et al. (2001). It mimics the musical improvisation process in which a group of musicians play the pitches of their musical instruments together seeking a pleasing harmony as determined by an audio-aesthetic standard. It is considered a population-based algorithm with local search-based aspects (Lee et al. 2005). This algorithm has an interesting feature that differentiates it from the other metaheuristics: it iteratively explores the search space by combining multi-search space regions to visit a single search space region. We have to recall that, through the recombination and randomness, the harmony search algorithm iteratively recombines the characteristics of many solutions in order to make one solution. It is able to fine tune this solution to which the algorithm converges using neighborhood structures. Throughout the process recombination is represented by memory consideration, randomness by random consideration, and neighborhood structures by pitch adjustment. In the typical population-based methods, the search space is explored by moving from multi-search space regions to multi-search space regions and the local search-based methods explore the search space regions moving from a single region to another. As such, the harmony search algorithm has the advantage of combining key components of population-based and local search-based methods in a simple optimization model.

Harmony search algorithm is a stochastic search mechanism, simple in concepts, and no derivation information is required in the initial search (Lee et al. 2005). It has been successfully tailored to a wide variety of optimization problems such as travelling salesman problem (Geem et al. 2001); structural design (Lee and Geem 2004); water network design (Geem 2006); dam scheduling (Geem 2007b), sudoku game (Geem 2007a); music composition (Geem and Choi 2007), and many others as discussed by Ingram and Zhang (2009).

The main aim of this paper is twofold: firstly, applying the basic harmony search for the university course timetabling as an initial exploration to this algorithm in this domain; secondly, modifying the functionality of the basic harmony search to be even more efficient for the university course timetabling problem because optimization problems offer No Free Lunch (Wolpert and Macready 1997). Results show that the basic harmony search can tackle this problem intelligently and offers near optimal solutions while the modified harmony search offers high quality solutions when compared to the previous methods.

The remainder of this paper includes the following sections: Sect. 2 discusses the university course timetabling problem. Section 3 explains the fundamentals of harmony search algorithm. The application of harmony search algorithm to university course timetabling is the purpose of Sect. 4. Section 5 discusses the experimental results and compares them with those in the previous literature. In the final section, we present a conclusion and some possible future directions to our proposed methods.

2 The university course timetabling problem

The University Course Timetabling Problem (UCTP) version tackled in this paper was produced by Socha et al. (2002) and it can be described as follows:

- A set $\mathcal{C} = \{c_0, c_1, \dots, c_{N-1}\}$ of N courses, each of which contains certain students and needs particular features.
- A set $\mathcal{R} = \{r_0, r_1, \dots, r_{K-1}\}$ of K rooms, each of which has a seat capacity and contains specific features.
- A set $\mathcal{S} = \{s_0, s_1, \dots, s_{L-1}\}$ of L students, each of them assigned to one or more courses.
- A set $\mathcal{F} = \{f_0, f_1, \dots, f_{M-1}\}$ of M features.
- A set $\mathcal{T} = \{t_0, t_1, \dots, t_{\mathrm{P}-1}\}$ of P timeslots.

Furthermore, the set of problem instances produced by Socha et al. (2002) provide the following information:

- A vector $\boldsymbol{a}$ of room capacity where a_i is the capacity of room i, $i \in \mathcal{R}$.
- A Student-Course matrix $\mathbf{U}$ where $u_{i,j} = 1$ denotes the student i assigns course j, $u_{i,j} = 0$ otherwise, $i \in \mathcal{S}$ and $j \in \mathcal{C}$.
- A Room-Feature matrix $\mathbf{V}$ which is described as a room i has a feature j if and only if $v_{i,j} = 1$, $i \in \mathcal{R}$ and $j \in \mathcal{F}$.
- A Course-Feature matrix $\mathbf{W}$ means that a course i needs feature j if and only if $w_{i,j} = 1$, $i \in \mathcal{C}$ and $j \in \mathcal{F}$.

The following hard constraints must be satisfied:

- H1. *Students must not be double booked for courses.*
- H2. *Room size and features must be suitable for the assigned courses.*
- H3. *Rooms must not be double booked for courses.*

And the following soft constraints should be minimized:

- S1. *A student shall not have a class in the last slot of the day.*
- S2. *A student shall not have more than two classes in a row.*
- S3. *A student shall not have a single class in a day.*

The main objective of the context of UCTP is to produce a feasible solution where the violations of soft constraints are minimized. It is worth mentioning that the context of UCTP reflects the real course timetabling problem at Napier University in Edinburgh.

Originally, the context of UCTP used by Socha et al. (2002) was determined by the Metaheuristics Network (MN).[1] MN is a European commercial research project shared by five European institutions between 2000 to 2004 to investigate the efficiency of different metaheuristics on different combinatorial optimization problems.

The same context of UCTP was used in the first International Timetabling Competition.[2] Twenty data instances and three more hidden ones were constructed. Those data instances were proposed mainly to motivate the competitors to focus their attention on generating effective approaches for UCTP. In fact, those data instances observed soft constraints minimization rather than hard constraints fulfillment. Some works that have lately appeared used the same data instances to measure the efficiency of their approaches (Chiarandini et al. 2006; Lewis and Paechter 2004; Kostuch 2005; Burke et al. 2003b).

The combinatorial optimization problems are difficult to solve due to the complexity and size of the problem and also due to the university community which has increased rapidly in the last five decades. With that in mind, Lewis and Paechter (2005) constructed sixty hard data instances of the same UCTP context defined by MN to measure the capability of the Grouping Genetic Algorithm to find feasible timetables. Tuga et al. (2007) used the same data instances to evaluate the performance of Simulated Annealing with Kempe Chain to find feasible timetables.

The post enrollment course timetabling problem (Lewis et al. 2007) was tracked on the Second International Timetabling Competition (ITC-2007) (McCollum et al. 2009). This is similar to the UCTP context of MN with slight differences: in ITC-2007, two more hard constraints were addressed. The twenty one problem instances constructed for this track tackled different sizes and complexity, and the *distance to feasibility* is considered to be another measurement for the quality of the solutions. The term distance to feasibility refers to the number of courses that are not scheduled in the timetable in which the number of students within each unscheduled course is a factor for evaluation.

3 Fundamentals of the harmony search algorithm

The following is a detailed explanation of the basics of the harmony search algorithm (HSA) and its relation to the musical context (Lee and Geem 2004, 2005).

3.1 Optimization in musical context

Before any explanation, it is worth delving into Table 1 which shows the relationship or equivalences between the optimization terms and the musical context. Figure 1 shows the analogy between the music improvisation process and optimization process. In musical improvisation, a group of musicians improvise the pitches of their musical instruments, practice after practice, seeking for a pleasing harmony as determined by an audio-aesthetic standard. Initially, each musician improvises any pitch from the possible pitch range which will finally lead all musicians to create a fresh harmony. That fresh harmony is estimated by an

[1] Metaheuristics Network official website http://www.metaheuristics.net/ (27-Sep-2009).

[2] First International Timetabling Competition was organized by Metaheuristics Network members and was sponsored by PATAT. The official website is http://www.idsia.ch/Files/ttcomp2002/ (27-Sep-2009).

Table 1 The optimization terms in the musical context

Musical terms		Optimization terms
Improvisation	⟷	Generation or construction
Harmony	⟷	Solution vector
Musician	⟷	Decision variable
Pitch	⟷	Value
Pitch range	⟷	Value range
Audio-aesthetic standard	⟷	Objective function
Practice	⟷	Iteration
Pleasing harmony	⟷	(Near-) optimal solution

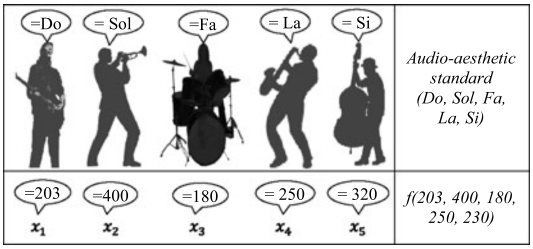
=Do
= Sol
=Fa
= La
= Si
Audio-aesthetic standard (Do, Sol, Fa, La, Si)
=203
=400
=180
= 250
= 320
x_1
x_2
x_3
x_4
x_5
f(203, 400, 180, 250, 230)

Fig. 1 Analogy between music improvisation and optimization process

audio-aesthetic standard: if it is good (i.e., involves better pitches than the preferable pitches in musicians' memory), the musicians retain the good pitches in their memory instead of those included within the worst harmony stored earlier for using them in the next practice. *Practice after practice*, the good pitches are stored in the musicians' memory which give them a chance to produce a pleasing harmony in their following practices.

This can be translated into optimization process as follows: a set of decision variables is assigned with values, iteration by iteration, seeking for a '*good enough*' solution as evaluated by an objective function. Initially, each decision variable is assigned by any value from its possible range which will finally lead all decision variables to create a new solution vector. That solution vector is evaluated by an objective function: if it is good (i.e., involves better values than experience values stored in the memory), the decision variables will store the good values in their memory instead of those included within the worst solution vector stored earlier for using them in the next iterations. *Iteration by iteration*, the good values for each decision variable will be stored in the memory giving them a chance to produce a better solution in the following iterations.

When each musician improvises a pitch from his musical instrument, he has three options: (i) improvising any pitch from his memory (ii) modifying a pitch which exists in the memory, or (iii) improvising any pitch from the possible pitch range. In optimization, each value of any decision variable is decided according to one of the following options (i) assigning a value stored in the memory. (ii) modifying a value which exists in the memory, or (iii) assigning a value from its feasible range. Geem et al. (2001) formalized these three options into three operators: memory consideration, pitch adjustment, and random consideration. These operators are controlled by two parameters named Harmony Memory Consideration Rate (HMCR) and Pitch Adjustment Rate (PAR) (these will be discussed in more detail in Sect. 3.2).

Figure 2 shows the harmony memory structure which is the core part of the improvisation process. Consider musical instruments of five musicians on the Jazz bandstand as

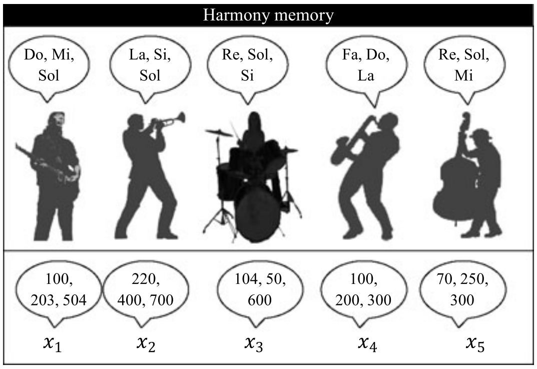

Fig. 2 The harmony memory structure

follows: Guitarist, Trumpeter, Drummer, Saxophonist, and Double bassist. There are sets of preferable pitches in their memory, that is Guitarist: $\{Do, Mi, Sol\}$; Trumpeter: $\{La, Si, Sol\}$; Drummer: $\{Re, Sol, Si\}$; Saxophonist: $\{Fa, Do, La\}$; Double bassist: $\{Re, Sol, Mi\}$. Assume in a practice if Guitarist randomly improvises $\{Do\}$ from his memory, Trumpeter improvises $\{Sol\}$ from his memory, Drummer adjusts $\{Re\}$ from his memory to come up with $\{Fa\}$, Saxophonist improvises $\{La\}$ from his memory, and Double bassist improvises $\{Si\}$ from the available range $\{Do, Re, Mi, Fa, Sol, Si\}$. All these pitches together form a fresh harmony (Do, Sol, Fa, La, Si) which is estimated by an audio-aesthetic standard. If the fresh harmony is better than the worst harmony stored in the harmony memory, the harmony memory will be updated by replacing the worst harmony with the fresh one. This process will be repeated to obtain a pleasing harmony.

The situation is similar in real optimization: consider five decision variables, each of which has stored experience values in harmony memory as follows, $x_1 : \{100, 203, 504\}$; $x_2 : \{220, 400, 700\}$; $x_3 : \{104, 50, 600\}$; $x_4 : \{100, 200, 300\}$; $x_5 : \{70, 250, 300\}$. Assume in an iteration if x_1 is assigned with 203 from its memory; x_2 is assigned with 400 from its memory; x_3 is adjusted from the value 104 stored in its memory to be 180; x_4 is adjusted from the value 200 stored in its memory to be 250; x_5 is assigned with 320 from its feasible range $x_3 \in [0, 600]$. A new solution vector $(203, 400, 180, 250, 320)$ is created and evaluated by an objective function. If the solution is better than the worst solution stored in the harmony memory, it is adopted whereas the worst solution is excluded. This process will be repeated time and again to find a (near) optimal solution.

3.2 The basic harmony search algorithm

Algorithm 1 shows the pseudo-code of the basic HSA with five main steps that will be described in the following:

Step 1. **Initialize the problem and HSA parameters**. Suppose that the discrete optimization problem is modeled as in (1).

$$\min\{f(\boldsymbol{x})|\boldsymbol{x} \in \mathbf{X}\}, \quad \text{Subject to} \quad g(\boldsymbol{x}) < 0 \quad \text{and} \quad h(\boldsymbol{x}) = 0, \tag{1}$$

where $f(\boldsymbol{x})$ is the objective function; $\boldsymbol{x} = \{x_i | i = 1, \dots, N\}$ is the set of each decision variable. $\mathbf{X} = \{\boldsymbol{X}_i | i = 1, \dots, N\}$ is the possible value range for each decision variable, where $\boldsymbol{X}_i = \{v_{i,1}, v_{i,2}, \dots, v_{i,K_i}\}$. N is the number of decision variables, and K_i is the number of values for each decision variable x_i. $g(\boldsymbol{x})$ are inequality constraint functions and $h(\boldsymbol{x})$ are equality constraint functions. The parameters of the HSA required to solve the optimization

Algorithm 1 The basic harmony search algorithm

```
1. STEP1. Initialize the problem and HSA parameters
     Input data. The data instance of the optimization problem and the HSA parameters
                 (HMCR, PAR, NI, HMS).
2. STEP2. Initialize the harmony memory
     Construct the vectors of the harmony memory, $\mathbf{HM} = \{x^1, x^2, \dots, x^{\text{HMS}}\}$
     Recognize the worst vector in HM, $x^{worst} \in \{x^1, x^2, \dots, x^{\text{HMS}}\}$
3. STEP3. Improvise a new harmony
     $\boldsymbol{x}' = \phi$ // new harmony vector
      for $i = 1, \dots, N$ do      // $N$ is the number of decision variables.
        if $(U(0,1) \le \text{HMCR})$ then     // $U$ is a uniform random number generator.
          begin
          $x_i' \in \{x_i^1, x_i^2, \dots, x_i^{\text{HMS}}\}$      {* memory consideration *}
          if $(U(0,1) \le \text{PAR})$ then
            $x_i' = v_{i,k\pm m}$      // $x_i' = v_{i,k}$      {*pitch adjustment *}
          end
        else
          $x_i' \in \boldsymbol{X}_i$          {* random consideration *}
        end if
      end for
4. STEP4. Update the harmony memory (HM)
     if $(f(\boldsymbol{x}') < f(\boldsymbol{x}^{worst}))$ then
       Include $\boldsymbol{x}'$ to the HM.
       Exclude $\boldsymbol{x}^{worst}$ from HM.
5. STEP5. Check the stop criterion
     while (not termination criterion is specified by NI)
       Repeat STEP3 and STEP4
```

problem are also specified in this step: the HMCR; the Harmony Memory Size (HMS) similar to population size in Genetic Algorithm; the PAR; and the Number of Improvisations (NI) or the number of iterations. Note that the HMCR and PAR are the two parameters responsible for the improvisation process. These parameters will be explained in more detail in the following steps.

***Step 2*. Initialize the harmony memory**. The harmony memory (HM) is a memory location which contains sets of solution vectors which are determined by HMS (see (2)). In this step, these vectors are randomly (or heuristically) constructed and stored to the HM based on the value of the objective function.

$$\mathbf{HM} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_N^1 \\ x_1^2 & x_2^2 & \cdots & x_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{\text{HMS}} & x_2^{\text{HMS}} & \cdots & x_N^{\text{HMS}} \end{bmatrix}. \tag{2}$$

***Step 3*. Improvise a new harmony**. In this step, the HSA will generate (improvise) a new harmony vector from scratch, $\boldsymbol{x}' = (x_1', x_2', \dots, x_N')$, based on three operators: (1) memory consideration, (2) random consideration, and (3) pitch adjustment.

Memory consideration. In memory consideration, the value of the first decision variable x'_1 is randomly assigned from the historical values, $\{x_1^1, x_1^2, \ldots, x_1^{\text{HMS}}\}$, stored in HM vectors. Values of the other decision variables, $(x'_2, x'_3, \ldots, x'_N)$, are sequentially assigned in the same manner with probability (w.p.) HMCR where $0 \leq \text{HMCR} \leq 1$. This operator acts similar to the recombination operator in other population-based methods and is a good source of exploitation (Yang 2009).

Random consideration. Decision variables that are not assigned with values according to memory consideration are randomly assigned according to their possible range by random consideration with a probability of (1-HMCR) as in (3).

$$x'_i \leftarrow \begin{cases} x'_i \in \{x_i^1, x_i^2, \ldots, x_i^{\text{HMS}}\} & \text{w.p. HMCR} \\ x'_i \in \boldsymbol{X}_i & \text{w.p. 1-HMCR.} \end{cases} \tag{3}$$

Random consideration is functionally similar to the mutation operator in Genetic Algorithm which is the source of global exploration in HSA (Yang 2009). The HMCR parameter is the probability of assigning one value of a decision variable, x'_i, based on historical values stored in the HM. For instance, if HMCR = 0.90, this means that the probability of assigning the value of each decision variable from historical values stored in the HM vectors is 90%, and the value of each decision variable is assigned from its possible value range with the probability of 10%.

Pitch adjustment. Each decision variable x'_i of a new harmony vector, $\boldsymbol{x}' = (x'_1, x'_2, x'_3, \ldots, x'_N)$, that has been assigned a value by memory considerations is examined for whether or not it should be pitch adjusted with the probability of PAR where $0 \leq PAR \leq 1$ as in (4).

$$\text{Pitch adjusting decision for } x'_i \leftarrow \begin{cases} \text{Yes} & \text{w.p. PAR} \\ \text{No} & \text{w.p. 1-PAR.} \end{cases} \tag{4}$$

A PAR of 0.10 means that the HSA modifies the existing value of decision variables assigned by memory consideration with a probability of (PAR × HMCR) while the other values of decision variables assigned by memory consideration do not change. If the pitch adjustment decision for x'_i is Yes, the value of x'_i is modified to its neighboring value as follows:

$$x'_i(k) = v_{i,k\pm m}, \tag{5}$$

where x'_i is assigned with value $v_{i,k}$, that is, the kth element in $\boldsymbol{X}_i$. m is the neighboring index, $m \in \mathbb{Z}$. Equation (6) summarizes the improvisation process

$$x'_i \leftarrow \begin{cases} x'_i \in \{x_i^1, x_i^2, \ldots, x_i^{\text{HMS}}\} & \text{w.p. HMCR} \\ x'_i = v_{i,k\pm m} & \text{w.p. HMCR} \times \text{PAR} \\ x'_i \in \boldsymbol{X}_i & \text{w.p. 1-HMCR.} \end{cases} \tag{6}$$

***Step 4.* Update the harmony memory**. If the new harmony vector, $\boldsymbol{x}' = (x'_1, x'_2, \ldots, x'_N)$, is better than the worst harmony stored in HM in terms of the objective function value, the new harmony vector is included to the HM, and the worst harmony vector is excluded from the HM.

***Step 5.* Check the stop criterion**. Steps 3 and 4 of HSA are repeated until the stop criterion (maximum number of improvisations) is met. This is specified by NI parameter.

Fig. 3 The location matrix which shows each location with its room-timeslot pair. For example, location 0 denotes the room-timeslot pair (0, 0); location 1 denotes the room-timeslot pair (0, 1); etc.

	t_0	t_1	$\dots$	t_{P-1}
r_0	0	1	$\dots$	$P-1$
r_1	P	$P+1$	$\dots$	$2P-1$
r_2	$2P$	$2P+1$	$\dots$	$3P-1$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
r_{K-1}	$(K-1)P$	$(K-1)P+1$	$\dots$	$KP-1$

4 The harmony search algorithm for UCTP

In order to choose a suitable timetable representation for the HSA, the timetable solution is represented by a vector of courses $\boldsymbol{x} = (x_1, x_2, \dots, x_N)$, each course must be scheduled in a feasible location; each location denotes a unique pair of room-timeslot. Each course, x_i, is to be scheduled in a feasible location within the range between $[0, K \times P - 1]$, where K and P, as pointed out earlier, is the number of rooms and timeslots consecutively (see the location matrix in Fig. 3). For example, in the medium problem instances established by Socha et al. (2002), the number of courses $N = 400$, the number of rooms $K = 10$ and the number of timeslots $P = 45$, the possible locations of each course, x_i, is within the range between 0 to 449. The HSA interprets the location of each course, x_i, as in (7):

$$x_i = j \times P + m. \tag{7}$$

This means that course x_i is scheduled in timeslot t_m at room r_j, j is the room index and m is the timeslot index. For example, let $\boldsymbol{x} = (449, 21, 102, \dots, 0)$ be a feasible and complete timetable. The HSA interprets the solution as follows: course x_1 is scheduled in location 449, in timeslot index 44 at room index 9; course x_2 is scheduled in location 21, in timeslot index 21 at room index 0; course x_3 is scheduled in location 102, in timeslot index 12 at room index 2; ...; course x_N is scheduled in location 0, in timeslot index 0 at room index 0. In practice, the timeslot index can be extracted form the location of the course x_i as in (8) and the room index can be extracted from the same location as in (9).

$$t_m = x_i \mod P, \tag{8}$$

$$r_j = \left\lfloor \frac{x_i}{P} \right\rfloor. \tag{9}$$

This solution representation directly satisfies the H3 hard constraint. Practically, the following data structures are used to build a university course timetabling solution:

- *Conflict matrix*: is a matrix $\mathbf{B}$ of size $N \times N$ where $b_{i,j}$ = the number of students sharing courses i and j. This matrix is used to deal with the H1 hard constraint.
- *Course room matrix*: is a binary matrix $\mathbf{D}$ of size $N \times K$ where $d_{i,j}$ contains either 1 if and only if course i and room j is compatible with both aspects of size and features or 0 otherwise. This matrix is used to deal with the H2 hard constraint.
- *Course position matrix*: is a binary matrix $\mathbf{Q}$ of size $N \times$ HMS where $q_{i,j}$ changes iteratively during the improvisation process (STEP 3 of HSA) which contains either $q_{i,j} = 1$ if and only if a course i has a feasible location in the solution j that is stored in HM to be scheduled in a new harmony solution or $q_{i,j} = 0$ otherwise. This matrix is initialized by 1 at the beginning of the improvisation process. It is also updated when a course is

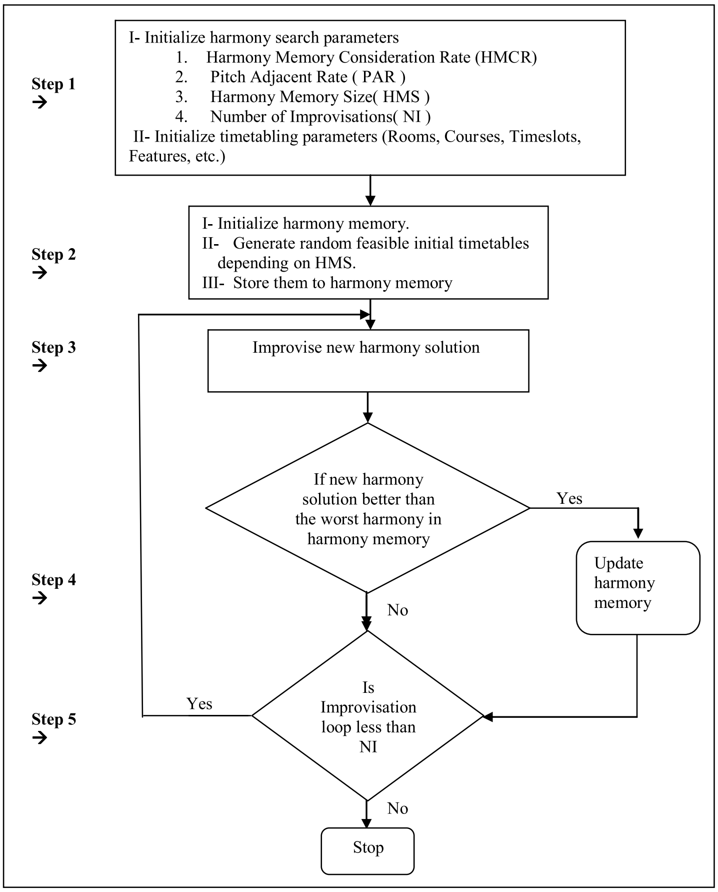
Step 1 →
I- Initialize harmony search parameters
1. Harmony Memory Consideration Rate (HMCR)
2. Pitch Adjacent Rate (PAR)
3. Harmony Memory Size(HMS)
4. Number of Improvisations(NI)
II- Initialize timetabling parameters (Rooms, Courses, Timeslots, Features, etc.)
Step 2 →
I- Initialize harmony memory.
II- Generate random feasible initial timetables depending on HMS.
III- Store them to harmony memory
Step 3 →
Improvise new harmony solution
Step 4 →
If new harmony solution better than the worst harmony in harmony memory
Yes
No
Update harmony memory
Step 5 →
Is Improvisation loop less than NI
Yes
No
Stop

Fig. 4 A harmony search algorithm for UCTP

scheduled out of memory consideration or random consideration, or adjusted by pitch adjustment operator.

Figure 4 describes the steps of HSA with application to the UCTP. It has to be borne in mind that this paper considers the feasible search space region. Therefore, some of the HSA steps and operators had to be modified to preserve the feasibility.

4.1 Initialize the HSA and UCTP parameters

Within UCTP, the parameters are extracted from the problem instances such as set of courses $\mathcal{C}$, set of rooms $\mathcal{R}$, set of timeslots $\mathcal{P}$, set of features $\mathcal{F}$, set of students $\mathcal{S}$, Room-Size vector $\boldsymbol{a}$, Student-Course matrix **U**, Room-Feature matrix **V**, and Course-Feature matrix **W**. These parameters are described in Sect. 2. Furthermore, in this step, the HSA builds the Conflict matrix **B** and Course room matrix **D**.

It has to be recalled that the main decision variables of UCTP are the *courses*; the location (*or* the room-timeslot pair) of each course might be changed during the search process of the HSA. The possible range of each course is the set of feasible locations available to it during the search.

Definition 1 The location l *is feasible* for course x_i to be scheduled in the timetable $\boldsymbol{x}$ if and only if the following conditions are met:

1. $x_j \neq l, \forall x_j \in \boldsymbol{x} \wedge i \neq j$,
2. $d_{i,\lfloor \frac{l}{P} \rfloor} = 1$,
3. $x_j \bmod P \neq l \bmod P, \forall x_j \in \boldsymbol{x} \wedge b_{i,j} > 0 \wedge i \neq j$.

The objective function described by Chiarandini et al. (2006) is utilized to evaluate the timetable solution, $\boldsymbol{x}$, as in (10):

$$f(\boldsymbol{x}) = \sum_{s=0}^{L-1} (f_1(\boldsymbol{x}, s) + f_2(\boldsymbol{x}, s) + f_3(\boldsymbol{x}, s)). \tag{10}$$

Where $f(\boldsymbol{x})$ is the objective function to evaluate the timetabling solution, $\boldsymbol{x}$. $f_1(\boldsymbol{x}, s)$, $f_2(\boldsymbol{x}, s)$ and $f_3(\boldsymbol{x}, s)$ describe the violation in the soft constraints S1, S2 and S3 consecutively for all students s where $s = 0 \dots L - 1$.

The HSA parameters described in Sect. 3 that are required to solve UCTP are also specified in this step namely, HMS, HMCR, PAR and NI.

4.2 Initialize the HM with random feasible timetable solutions

In Step 2, HSA constructs feasible timetabling solutions as determined by HMS. HM is filled with these solutions. See (2). The objective function value for all solutions in HM is maintained separately. Meanwhile, the solutions are increasingly sorted in HM according to their objective function value.

In the UCTP, a backtracking algorithm (Carter et al. 1996) and the proposed MultiSwap algorithm are applied to generate random HM solutions after assigning the courses by using the weighted largest degree (WLD) first heuristic method (Arani and Lofti 1989). This strategy ensures that all HM solutions are feasible. In WLD, the course with the largest number of conflicting students is scheduled first.

All courses that cannot be assigned to the timetable solution after the completion of the assignment process by the WLD heuristic method are entered to a list called the unscheduled list.

This list is then passed to a backtracking algorithm that will select each unscheduled course x_i from the unscheduled list and explore *all courses* in conflict using a Conflict matrix **B**. Those courses that share one or more students with course x_i are removed from the timetable solution and added to the unscheduled list again. After that, the backtracking algorithm attempts to assign feasible locations to all courses in the unscheduled list. This process is iterated several times until no further locations can be filled. Some courses may not be scheduled at the end of this process. In this case, the MultiSwap algorithm will be used.

The proposed MultiSwap algorithm shuffles the courses in different rooms within the same timeslot where the shuffling is performed consecutively at all timeslots. It is worth mentioning that there are two reasons why unscheduled courses cannot find feasible locations. Firstly, the appropriate rooms for the unscheduled courses are reserved by other courses. Secondly, the timeslots which contain courses share a student or more with the unscheduled course. Backtracking handles the second reason while MultiSwap tackles the first one. In MultiSwap algorithm, courses are taken from the same timeslot and shuffled to different suitable rooms using Course room matrix **D** in the hope of finding the appropriate rooms for the unscheduled courses.

Algorithm 2 Schematic pseudo-code of building HM solutions

```
for $i = 1, \dots, \text{HMS}$ do
    repeat
        $\boldsymbol{x}^i = \phi$
        WLD($\boldsymbol{x}^i$)
        while($\boldsymbol{x}^i$ is not complete || predefined iterations are not met) do
        begin
            Backtracking($\boldsymbol{x}^i$)
            MultiSwap($\boldsymbol{x}^i$)
        end
    until($\boldsymbol{x}^i$ is complete)
    store $\boldsymbol{x}^i$ in the HM
    calculate $f(\boldsymbol{x}^i)$
end for
```

If this process with predefined iterations cannot find a feasible solution, we propose to restart the whole process all over again. The schematic pseudo-code of building HM solutions is shown in Algorithm 2.

4.3 Improvise a new harmony solution

In Step 3, a new harmony timetabling solution, $\boldsymbol{x}' = (x_1', x_2', \dots, x_N')$, is generated from scratch based on three operators: (i) memory consideration, (ii) random consideration, (iii) pitch adjustment. The new harmony in this paper must be feasible and complete. In some iterations, the HSA operators may not improvise (generate) a complete timetable. In such cases, the repair process has to take over. Algorithm 3 shows the pseudo-code for improvising a new harmony solution.

For UCTP or generally for any timetabling problems, constructing a feasible solution (in our case a new harmony solution) is a crucial task. As such, the common idea to reserve the feasibility for the timetabling solution is to order the courses according to how difficult they are to be scheduled in the new harmony solution (i.e., graph coloring heuristic methods). Note that the improvisation step of basic HSA (See Algorithm 1, STEP 3) selects the decision variables to be assigned in the new harmony solution sequentially, starting from x_1' until x_N'. However, it is difficult to sequentially find feasible locations for all courses in the new harmony solution. With analogy to the ordering priority of the largest saturation degree (Brélaz 1979) where the courses must be ordered iteratively one by one based on the assigning difficulties during the construction process, the proposed *smallest position algorithm* is responsible for selecting courses with the least feasible locations in HM solutions by using the *Course position matrix*. Formally, let $n_k = \sum_{j=1}^{\text{HMS}} q_{k,j}$ be the total number of feasible location of course x_k, the smallest position algorithm selects course x_i' where

$$i = \arg \min_{k=1,\dots,N} n_k$$

If there is more than one course at each iteration with the same least feasible locations, the proposed algorithm selects one course depending on the WLD heuristic.

Algorithm 3 Improvise a new harmony solution

```
begin
  $x' = \phi$
  for $j = 1 \dots N$ do
    begin
     $x'_i$ = Smallest_positions()
    if ($U(0, 1) \le$ HMCR) then
      begin
       $x'_i = x^j_i$, where ($x^j_i \in B^{best}$)
       $rnd = U(0, 1)$
        if ($rnd \le PAR1$) then
          Pitch adjustment Move($x'_i$)
        elseif ($rnd \le PAR2$) then
          Pitch adjustment Swap-location($x'_i$)
        elseif ($rnd \le PAR3$) then
          Pitch adjustment Swap-timeslot($x'_i$)
      end
    else
        $x'_i \in \mathcal{Q}_i$, where $\mathcal{Q}_i = \{l | l$ is feasible for $x'_i, l \in [0, P \times K - 1]\}$
    end
   repair_process($x'$)
end
```

4.3.1 *Memory consideration*

Basic memory consideration: The basic memory consideration selects feasible locations of the courses to be scheduled in the new harmony solution, $\boldsymbol{x}' = (x'_1, x'_2, \dots, x'_{\mathrm{N}})$, from the solutions stored in HM with the probability of HMCR. Formally, let course x'_i be specified by the smallest position algorithm, let set $\mathcal{H}_i = \{x^j_i | q_{i,j} = 1, \forall j \in [1, \mathrm{HMS}]\}$ where ($i = 1 \dots \mathrm{N}$) and $q_{i,j} \in \mathbf{Q}$. The location of course x'_i will be selected *randomly* from set $\mathcal{H}_i$ with probability of HMCR.

Modified memory consideration: In UCTP, we modify the functionality of the basic memory consideration operator so as to always mimic the best solutions so far stored in HM that have feasible locations for all courses, such that,

$$B^{best} = \left\{ x^j_i | j = \arg \min_{k \text{ s.t. } x^k_i \in \mathcal{H}_i} f(\boldsymbol{x}^k) \right\}.$$

In other words, the location of course x'_i is selected from the best solution, so far stored in HM, that has a feasible location for x'_i such that $x'_i = x^j_i$ where $x^j_i \in B^{best}$ with probability HMCR. This idea stems from the analogy of Particle Swarm Optimization (PSO) (Kennedy and Eberhart 1995) where a swarm of individuals (called particles) explore the search space. Each particle is a candidate solution. It is drawn back to its best position and to the best position in the whole swarm once a new best particle is found.

4.3.2 *Random consideration*

The remaining courses that have not been scheduled by memory consideration will select any feasible location available to be scheduled in the new harmony solution with probability (1-HMCR). Formally, let course x_i' be selected by the smallest position algorithm to be scheduled in the new harmony solution, let set $\mathcal{Q}_i = \{l | l \text{ is feasible for } x_i', l \in [0, P \times K - 1]\}$. The course x_i' will be scheduled in any feasible location in set $\mathcal{Q}_i$ with probability of (1-HMCR).

4.3.3 Pitch adjustment

In the basic HSA, the pitch adjustment operator is mainly designed for mathematical and engineering optimization problems where the values of the examined decision variables that meet the PAR probability are replaced by the neighboring values by means of modifying the decision variable by m (see (5)). This does not seem practical in UCTP.

For UCTP, we designed the pitch adjustment operator to work similar to neighborhood structures in local search-based methods as follows: we divide the pitch adjustment operator into three procedures. (i) The pitch adjustment *Move*, (ii) the pitch adjustment *Swap-location*, and (iii) the pitch adjustment *Swap-timeslot*. Each course x_i' scheduled out of memory consideration is examined as to whether it should be pitch adjusted with probability of PAR where $0 \leq \text{PAR} \leq 1$. The PAR in our study is divided into three parameters PAR1, PAR2 and PAR3, each of which controls the pitch adjustment procedure as in (11):

$$\text{Adjust the value of } x_i' \leftarrow \begin{cases} \text{Move} & 0 < U(0,1) \leq \text{PAR1} \\ \text{Swap-location} & \text{PAR1} < U(0,1) \leq \text{PAR2} \\ \text{Swap-timeslot} & \text{PAR2} < U(0,1) \leq \text{PAR3} \\ \text{do nothing} & \text{PAR3} < U(0,1) \leq 1. \end{cases} \tag{11}$$

It is worth emphasizing again that only the courses that are scheduled according to memory consideration are examined by pitch adjustment procedures to determine the need for such adjustment. The courses that are scheduled out of random consideration are not examined by pitch adjustment procedures. This can be seen from Algorithm 3 where the pitch adjustment procedures run within the memory consideration operator. This is discussed by Yang (2009) who held that the pitch adjustment in the musical context allows the musicians to explore the preferable pitches stored in their memory while the pitch adjustment in the optimization context helps the HSA to locally explore the search space region of each decision variable. The three proposed pitch adjustment procedures are designed to work as follows:

– *Pitch adjustment Move*. A course x_i' that meets probability PAR1 is randomly *moved* to any free feasible location in the new harmony solution.
– *Pitch adjustment Swap-location*. A course x_i' that meets the range of probability PAR1 and PAR2 is randomly swapped with another course (e.g., x_j') that has already been scheduled in the new harmony while maintaining the feasibility.
– *Pitch adjustment Swap-timeslot*. A course x_i' that meets the range of probability PAR2 and PAR3 is adjusted as follows: (i) select all courses that have the same timeslot (e.g., t_j) as course x_i'; (ii) select a timeslot at random (e.g., t_k); (iii) simply swap all the courses in the timeslot t_j with all the courses in the other timeslot t_k without changing the rooms. Formally, let x_i' be scheduled by memory consideration and examined for pitch adjustment by Pitch adjustment Swap-timeslot. Let the set $\mathcal{A} = \{x_j | x_j \bmod P = x_i' \bmod P, \forall j \in [1, N]\}$ contain all courses scheduled in a new harmony solution $\boldsymbol{x}'$ having the same timeslot as

the course x_i'. Let the set $\mathcal{B} = \{x_b | x_b \bmod P = t_k, \forall b \in [1, N]\}$ contain all courses scheduled in $\boldsymbol{x}'$ that have the same randomly selected timeslot t_k where $(t_k \neq x_i' \bmod P)$. Simply $\forall x_j \in \mathcal{A}, x_j = \lfloor \frac{x_j}{P} \rfloor \times P + t_k$ and $\forall x_b \in \mathcal{B}, x_b = \lfloor \frac{x_b}{P} \rfloor \times P + x_i' \bmod P$.

Note that the pitch adjustment in the basic harmony search accepts the adjustments of all examined decision variables randomly (i.e., as a random walk in the search space), without checking if these adjustments will not negatively affect the objective function value. In the case of the UCTP, the number of decision variables is considerable, and the random acceptance rule may lead to undesirable diversity. Therefore, a possible way to manage this operator is to accept the adjustments done by any pitch adjustment procedure mentioned above, on the condition that the objective function value of the new harmony solution is not negatively affected (i.e., side walk and first improvement acceptance rule). In this paper, we modified the pitch adjustment procedures to accept the adjustments performed by (11), *if and only if* the objective function value of the new harmony solution is not negatively affected. Contrary to what (Yang 2009) explained as the pitch adjustment operator being used for local exploration, our study has modified such an operator to be used for local exploitation.

4.3.4 Repair process

During the improvisation process of a new harmony solution, some courses that were supposed to be scheduled based on the operators of memory consideration or random consideration were not able to find feasible locations in the new harmony solution. This occurs when the HMS is small and the size of the timetable instance is medium or large. The process of producing a feasible new harmony needs a repair process to schedule these unscheduled courses. It is indeed difficult to design an effective repair process that changes the new harmony solution from an incomplete state to a complete one without affecting the optimization nature of the harmony search. Thus, the repair process used here is based on a one-level backtracking process. In other words, the repair process is iterative during which the following operations are performed at each iteration:

1. Select an unscheduled course x_i.
2. Find all feasible locations for the unscheduled course x_i which is currently occupied by other courses in the new harmony solution.
3. Greedily select the best feasible location (e.g., l) for the unscheduled course x_i in terms of the value of the objective function.
4. Delete the course (e.g., x_j) that held the feasible location l and add it to the unscheduled list.
5. Schedule the unscheduled course x_i to new harmony solution in the feasible location l and remove it from the unscheduled list.

The repair process then attempts to find new feasible locations for the new unscheduled courses. If the repair process with the predefined iterations cannot find a complete feasible timetable, the improvisation process of new harmony is restarted with a new random seed.

The main difference between the backtracking algorithm used in Sect. 4.2 and the one-level backtracking used in this section is that the backtracking algorithm removes all courses conflicting with the unscheduled courses. Thus, the number of courses is probably high, which may affect the efficiency of the HSA operators if the same algorithm is used to repair the new harmony solution. The one-level backtracking removes one course at a time from

Table 2 The characteristics of each class of Socha benchmark

Class	Small	Medium	Large
Number of events	100	400	400
Number of rooms	5	10	10
Number of features	5	5	10
Number of timeslots	45	45	45
Approximate features per room	3	3	5
Percentage of the feature use	70%	80%	90%
Number of students	80	200	400
Maximum events per student	20	20	20
Maximum students per event	20	50	100

a new harmony solution based on the objective function. Evidently then, the efficiency of HSA operators mentioned above are not highly affected.

Finally, we apply for UCTP the same functionality of Steps 4 and 5 discussed in Sect. 3.2.

5 Experimental results

The performance of our basic HSA and modified harmony search algorithm (MHSA) for UCTP are evaluated in this section. The basic HSA used the functionally of HSA operators as appeared in Lee and Geem (2005), where it used a basic memory consideration and pitch adjustment with random walk as an acceptance rule. The MHSA changes the functionality of the basic HSA as described in Sect. 4, where it used a modified memory consideration and pitch adjustment with side walk and first improvement acceptance rule. The proposed methods are coded in Microsoft Visual C++ 6 under Windows XP platform on an Intel 2 GHz Core 2 Quad processor with 4 GB of RAM.

5.1 The problem instances

The UCTP data used in the experiments in this study are freely available,[3] prepared by Socha et al. (2002). For the purposes of our study, we call them the ‘Socha benchmark’. The 11 problem instances, which are grouped into five small problem instances, five medium problem instances and one large problem instance, have different levels of complexity and various sizes, as shown in Table 2. The solution to all problem instances must satisfy the defined hard constraints stated in Sect. 2. Furthermore, the solution cost is measured by the defined soft constraint violations as described in (10).

5.2 Empirical study of the impact of different parameter settings on convergence behavior of MHSA

The main aim of this section is to study the features of MHSA operators during the search process on different settings of five parameters (i.e., HMS, HMCR, PAR1, PAR2, and

[3] See http://iridia.ulb.ac.be/~msampels/tt.data/ (27-Sep-2009).

Table 3 Different MHSA convergence scenarios

HMS	HMCR	PAR1	PAR2	PAR3	Scenario No.
1	100%	0%	0%	0%	1
		2%	4%	6%	2
	99%	0%	0%	0%	3
		2%	4%	6%	4
		20%	40%	60%	5
10	99%	0%	0%	0%	6
		2%	4%	6%	7
		20%	40%	60%	8
50	100%	0%	0%	0%	9
		2%	4%	6%	10
	99%	0%	0%	0%	11
		2%	4%	6%	12
		20%	40%	60%	13

PAR3). Our discussion takes in consideration the exploration and exploitation search aspects. Generally, any successful metaheuristic can explore the not-yet-visited search space regions when it is necessary (i.e., *exploration*). It can also make use of the already visited search space regions (i.e., *exploitation*). Exploration and exploitation are contradictory and thus a suitable balance between them should be made to reach a high quality solution.

In particular, we designed 13 convergence scenarios at different parameter settings to show the convergence behavior of the proposed MHSA method as shown in Table 3. Each designed scenario was run 10 times with iteration numbers fixed to 100,000 for all runs. We experimented each scenario on the Socha benchmarks.

In Table 4, the best, average, worst, and standard deviation of the solution costs (see (10)) are recorded together with the computational time for each scenario. The best result among all scenarios on a particular problem instance is highlighted in bold.

Scenarios 1 to 5 are meant to show the behavior of the MHSA when the HMS $= 1$. In these scenarios, the MHSA behaves similar to local search-based methods. Scenario 1 shows that the MHSA does not have exploration and exploitation source. In each iteration, a new harmony solution is constructed by inheriting the locations of courses from a single solution stored in HM. The locations of courses and new harmony cost do not change during each search. In Scenario 2, the MHSA works similar to the Iterative Local Search Algorithm with the three defined neighborhood structures. The new harmony solution is always accepted if its cost is better than or equal to the solution cost stored in HM. Here the MHSA is concerned with exploitation rather than exploration which causes it to easily get stuck in local optima.

It can be observed from Scenario 3 that the MHSA behaves like Iterative Local Search but without neighborhood structure definition (i.e., pitch adjustment procedures). The ability of this scenario to improve the new harmony solution is based on constructing a new harmony solution in each iteration that selects most of the locations of the courses from a single solution stored in HM and few locations of the other courses randomly selected. *We believe that this scenario may lead to new research trends for adapting the ability of existing local search-based methods in the exploration power discipline.*

To assess the above observation, in Scenario 4 the MHSA behaves like Iterative Local Search with three defined neighborhood structures and random consideration as an auxiliary

Table 4 MHSA convergence scenarios (Scenarios 1 to 13)

Dataset		Scen. 1	Scen. 2	Scen. 3	Scen. 4	Scen. 5	Scen. 6	Scen. 7
Small 1	best	178	4	7	**0**	**0**	5	1
	average	209.67	5.4	9.4	2.5	0.1	7.9	2.3
	worst	256	7	11	4	1	10	4
	std.dev.	32.6	1.34	1.42	1.17	0.31	2.33	1.15
	time(s)	45	160	70	155	750	93	178
Small 2	best	165	1	9	**0**	**0**	7	1
	average	221.1	2.6	10.4	2.5	1.1	8.7	2.4
	worst	270	4	12 4	2	1	1	4
	std.dev.	43.826	0.966	0.966	1.269	0.737	1.337	0.966
	time(s)	145	145	64	153	830	74	176
Small 3	best	198	4	7	3	**0**	9	2
	average	242.75	5.6	9.7	4.5	0.8	10.4	3
	worst	301	7	12	6	2	12	5
	std.dev.	35.098	0.966	1.418	1.080	0.788	1.074	1.054
	time(s)	38	155	53	126	870	84	195
Small 4	best	188	4	7	4	**0**	5	2
	average	235.4	5.3	9.6	5.8	1.2	6.3	3.8
	worst	265	7	12	8	3	8	5
	std.dev.	28.675	0.948	1.577	1.316	0.918	0.948	1.229
	time(s)	52	150	81	168	904	83	168
Small 5	best	196	**0**	4	**0**	**0**	1	**0**
	average	229.7	2.1	5.4	0.4	0	2.6	0.4
	worst	284	4	7	2	0	4	2
	std.dev.	28.075	1.911	1.074	0.699	0	1.173	0.699
	time(s)	42	140	45	155	760	85	172
Medium 1	best	735	220	273	191	169	277	207
	average	806.33	239.9	281.6	209.3	180.1	288.4	220.3
	worst	873	256	290	215	196	300	236
	std.dev.	51.090	13.682	5.125	8.193	8.672	8.235	8.680
	time(s)	850	2133	1129	2712	7645	1574	3185
Medium 2	best	712	238	251	182	161	235	168
	average	758.8	252.67	260.22	200.11	169.78	254	182.22
	worst	803	274	273	209	188	271	198
	std.dev.	36.437	12.531	8.714	9.293	8.899	13.564	11.997
	time(s)	860	2036	1204	2644	8566	1873	3455
Medium 3	best	679	214	290	190	177	273	209
	average	758.5	256.4	303.3	216.4	189.6	293.9	221.2
	worst	855	279	324	234	206	311	233
	std.dev.	72.633	19.224	10.853	12.946	9.143	13.219	8.791
	time(s)	889	2150	1316	2734	8259	1634	3177

Table 4 (*Continued*)

Dataset		Scen. 1	Scen. 2	Scen. 3	Scen. 4	Scen. 5	Scen. 6	Scen. 7
Medium 4	best	760	247	234	164	159	238	166
	average	787.33	260.3	255	180.1	173.4	253.1	178.1
	worst	813	275	268	198	187	276	189
	std.dev.	22.983	9.638	12.640	10.660	10.101	12.187	7.445
	time(s)	765	2054	1198	2577	7543	1544	3265
Medium 5	best	688	97	208	120	73	203	133
	average	759.5	114.6	216.8	135.5	94.5	222.6	143
	worst	834	130	225	157	108	237	156
	std.dev.	51.149	10.731	5.788	10.834	9.144	12.624	8.485
	time(s)	967	1930	1334	2756	8792	1769	3320
Large	best	1634	563	694	514	432	661	516
	average	1720.5	621.3	726.5	565.5	502.1	708.37	547.8
	worst	1812	666	770	655	579	771	582
	std.dev.	66.416	33.243	31.0385	42.675	40.355	33.866	25.646
	time(s)	1878	2955	2059	2959	13256	2533	3539

operator to diversify the new harmony solution. In each iteration, the MHSA selects few random locations to some courses through the improvisation process. Similarly, in Scenario 5, the MHSA behaves similar to Scenario 4, but there is a greater use of the pitch adjustment operator. Therefore, the MHSA is able to fine-tune the search space region of the new harmony solution more rigorously. Although the MHSA in Scenarios 4 and 5 is able to find a suitable trade off between exploration and exploitation, it still works as a local search-based method.

In the remaining Scenarios 6 to 13, The MHSA makes use of the strength of the modified memory consideration by means of selecting better locations of the most courses from the best solutions, so far stored in HM, to generate the new harmony solution. It can be seen from Scenarios 6 and 11 that the MHSA is able to generate the new harmony solution based on modified memory consideration and random consideration. However, MHSA does not concentrate on the search space region of the new harmony solution to which it converges. That is, the pitch adjustment procedures are not used. Although the MHSA is able to recognize the promising search space regions, it is not able to precisely fine-tune any of them.

Scenarios 7 and 12 are designed to show the ability of the MHSA to improvise a new harmony solution using the modified memory consideration and random consideration which recognize the promising search space region and pitch adjustment procedures to fine-tune. Scenarios 8 and 13 are likewise meant to show the ability of MHSA to rigorously fine-tune the search space region of a new harmony solution by using the pitch adjustment procedures with larger PAR values. Basically, in scenarios (7, 8, 12 and 13), the MHSA is able to strike a balance between exploration and exploitation in addition to its ability to scan many search space regions at the same time, thus leading to the most desired results from these scenarios (see Table 4).

Finally, Scenario 9 is set to show the ability of MHSA to improvise a new harmony solution based on modified memory consideration only. Since the HMCR $= 1$ (i.e., random consideration is not used), the MHSA concentrates on exploitation rather than exploration.

Table 4 (*Continued*)

Dataset		Scen. 8	Scen. 9	Scen. 10	Scen. 11	Scen. 12	Scen. 13
Small 1	best	**0**	88	1	6	2	**0**
	average	0.3	91.2	4.2	8	3.3	0
	worst	1	95	6	10	5	0
	std.dev.	0.483	2.936	1.619	1.490	1.159	0
	time(s)	873	301	353	328	387	1093
Small 2	best	**0**	97	2	5	1	**0**
	average	0.9	101.4	3.7	7.9	2.4	1
	worst	2	109	6	10	4	2
	std.dev.	0.737	4.452	1.337	1.595	0.966	0.816
	time(s)	858	275	325	284	365	982
Small 3	best	**0**	90	2	7	2	**0**
	average	1.2	97.4	3.9	9.3	3.6	0.4
	worst	2	105	6	11	5	2
	std.dev.	0.788	5.601	1.370	1.567	0.966	0.699
	time(s)	916	259	326	298	365	1065
Small 4	best	**0**	111	5	6	**0**	**0**
	average	0.8	120.7	6.5	7.2	2.9	0.3
	worst	2	133	8	10	4	2
	std.dev.	0.788	7.498	0.971	1.316	1.449	0.674
	time(s)	1012	297	312	302	343	1163
Small 5	best	**0**	95	**0**	3	**0**	**0**
	average	0.2	98	1.7	4	0	0
	worst	1	101	4	7	0	0
	std.dev.	0.421	2.108	1.702	1.333	0	0
	time(s)	958	286	342	312	356	1094
Medium 1	best	180	646	234	274	196	**168**
	average	191.1	659.7	254.3	282.7	210.4	179.7
	worst	204	675	266	290	227	200
	std.dev.	7.430	10.985	12.552	5.396	10.002	10.296
	time(s)	13456	3460	4938	3870	5281	17357
Medium 2	best	175	659	211	252	169	**160**
	average	182.67	666.11	234.89	269.22	187.56	178.67
	worst	190	676	257	288	200	188
	std.dev.	4.5	5.988	15.551	14.889	11.147	9.772
	time(s)	12578	3515	4947	3964	5232	18185
Medium 3	best	189	651	217	275	194	**176**
	average	205.1	680.8	236.3	283.8	210.3	182.8
	worst	220	718	254	296	222	196
	std.dev.	9.949	28.326	11.450	9.461	8.380	7.699
	time(s)	14672	3845	5019	3905	5316	18452

Table 4 (*Continued*)

Dataset		Scen. 8	Scen. 9	Scen. 10	Scen. 11	Scen. 12	Scen. 13
Medium 4	best	**144**	631	206	238	173	152
	average	153.4	653.9	219.9	255	186.1	166
	worst	161	689	229	278	204	177
	std.dev.	7.471	17.381	8.849	14.204	9.643	10.697
	time(s)	13655	3684	4874	3872	5143	17954
Medium 5	best	90	612	114	204	139	**71**
	average	106.7	628.2	130	221.4	144.8	80.2
	worst	113	641	144	244	150	92
	std.dev.	6.111	9.919	10.392	11.871	3.735	8.521
	time(s)	13786	3540	5112	4094	5364	18536
Large	best	468	1409	558	633	534	**417**
	average	530.7	1453.6	620.4	675.2	556.9	476.6
	worst	563	1513	655	712	605	530
	std.dev.	36.514	48.724	34.644	31.336	33.238	37.322
	time(s)	14865	4253	6322	4653	6848	23716

As such, the method is easily gets stuck in local optima. In Scenario 10, the MHSA is able to improve the new harmony solution based on modified memory consideration and pitch adjustment procedures. The search does not concentrate on exploration since random consideration is not used. Therefore, the MHSA might easily get stuck in the local minima.

In short, larger HMS allows the MHSA to explore multiple search space regions simultaneously. Also, the larger the PAR values are, the more rigorous is the fine-tuning of the search space region to which the MHSA converges. In addition, the larger HMCR, the less exploration and the greater exploitation. In UCTP, the value of HMCR should be large to avoid the large exploration and thus the algorithm will not behave like a pure random search. It is to be noted that in some problem instances the repair process may increase the diversity of the MHSA.

The computational time of MHSA is influenced by two factors: the HMS and PAR values. The larger the HMS and PAR values is, the longer the computational time. In other words, a large HMS means that the memory consideration requires more computational time to find feasible locations for each course to be scheduled in the new harmony solution from many solutions stored in HM. The larger PAR1, PAR2, and PAR3 values cause the MHSA to make considerable local changes using the pitch adjustment operator which increases computational time.

5.3 Comparing results between basic HSA and MHSA

This section discusses the results obtained by the basic HSA and MHSA. Scenarios 4, 7 and 12 (Table 3) are used to compare both methods where the number of iterations is fixed to 100,000. These scenarios are chosen because they use all available operators. Each scenario runs 10 times. In Table 5, the best, average, worst, and standard deviation of the solu-

Table 5 Comparison results between basic HSA and MHSA

Dataset		Basic HSA			MHSA		
		Scen. 4	Scen. 7	Scen. 12	Scen. 4	Scen. 7	Scen. 12
Small 1	best	5	5	3	**0**	1	2
	average	7.5	7.1	5	2.5	2.3	3.3
	worst	10	9	8	4	4	5
	std.dev.	1.433	1.197	1.632	1.178	1.159	1.159
	time(s)	66	115	269	155	178	387
Small 2	best	14	8	4	**0**	1	1
	average	18.5	10.2	6.3	2.5	2.4	2.4
	worst	22	15	9	4	4	4
	std.dev.	2.368	2.347	1.494	1.269	0.966	0.966
	time(s)	72	132	278	153	176	365
Small 3	best	10	8	**2**	3	**2**	**2**
	average	13.2	10.5	3.7	4.5	3	3.6
	worst	17	13	5	6	5	5
	std.dev.	1.932	1.58	1.059	1.080	1.054	0.966
	time(s)	69	125	261	126	195	365
Small 4	best	11	6	3	4	2	**0**
	average	13.2	7.3	3.4	5.8	3.8	2.9
	worst	16	9	5	8	5	4
	std.dev.	1.549	1.159	0.843	1.316	1.229	1.449
	time(s)	73	145	283	168	168	343
Small 5	best	5	3	1	**0**	**0**	**0**
	average	6.5	4	2.8	0.4	0.4	0
	worst	8	6	4	2	2	0
	std.dev.	1.080	1.054	1.032	0.699	0.699	0
	time(s)	82	155	301	155	172	356
Medium 1	best	296	314	308	**191**	207	196
	average	307.3	340	317	209.3	220.3	210.4
	worst	318	366	326	215	236	227
	std.dev.	8.602	36.769	12.727	8.192	8.680	10.002
	time(s)	1540	2469	4320	2712	3185	5281
Medium 2	best	248	278	236	182	**168**	169
	average	255.1	291.7	245.1	200.111	182.222	187.556
	worst	267	312	256	209	198	200
	std.dev.	5.087	10.242	6.573	9.293	11.997	11.147
	time(s)	1354	2351	4258	2644	3455	5232
Medium 3	best	312	308	255	**190**	209	194
	average	344.3	327	274.3	216.4	221.2	210.3
	worst	363	344	286	234	233	222
	std.dev.	13.960	11.803	11.294	12.946	8.791	8.380
	time(s)	1428	2532	4492	2734	3177	5316

Table 5 (*Continued*)

Dataset		Basic HSA			MHSA		
		Scen. 4	Scen. 7	Scen. 12	Scen. 4	Scen. 7	Scen. 12
Medium 4	best	275	253	231	**164**	166	173
	average	286.8	265.9	244.7	180.1	178.1	186.1
	worst	312	274	265	198	189	204
	std.dev.	10.881	7.093	10.371	10.660	7.445	9.643
	time(s)	1242	2531	3987	2577	3265	5143
Medium 5	best	251	221	207	**120**	133	139
	average	265.8	235.3	214.7	135.5	143	144.8
	worst	276	245	222	157	156	150
	std.dev.	7.375	7.930	4.945	10.834	8.485	3.735
	time(s)	1429	2423	4145	2756	3320	5364
Large	best	–	–	–	**514**	516	534
	average	–	–	–	565.5	547.8	556.9
	worst	–	–	–	655	582	605
	std.dev.	–	–	–	42.675	25.646	33.238
	time(s)	–	–	–	2959	3539	6848

tion costs for each scenario are recorded together with the computational time upon Socha benchmarks. The best results obtained from the experimented scenarios are highlighted in bold.

It is apparent from Table 5 that the basic HSA is able to find feasible solutions for small and medium problem instances but not for large ones. The MHSA is able to obtain feasible results for all Socha benchmarks. The solution costs of the obtained results from MHSA outperforms those obtained by basic HSA in all the scenarios.

It is worth mentioning that the basic HSA cannot converge to the optimal solution with larger PAR values, mainly because the number of random local changes in the new harmony will be large, leading to a high diversity. Thus, the basic HSA will behave like a pure random search. For this reason, the Scenarios 5, 8, and 13 are poor choices for the basic HSA.

It can be noted that the computational time needed for basic HSA is less than the computational time needed for MHSA when both methods use the same scenario. This is because the MHSA uses the objective function each time to accept the local changes on the new harmony solution while basic HSA does not do so.

Practically, we notice that the modified memory consideration improves the speed of convergence of the basic HSA as well as reduces the selection pressure of the basic memory consideration operator. This modification basically helps the MHSA to configure a high quality new harmony solution at each run similar to the quality of the best solutions so far stored in HM. Also, the MHSA is able to find a feasible new harmony for large timetabling problem instances.

5.4 Comparison with previous works

The results are compared to those in the literature that used the same Socha benchmarks abbreviated in Table 4 as follows:

Table 6 Comparison results with the previous methods

	Small 1	Small 2	Small 3	Small 4	Small 5	Medium 1	Medium 2	Medium 3	Medium 4	Medium 5	Large
basic HSA(best)	3	4	2	3	1	296	236	255	231	207	–
MHSA(best)	**0**	**0**	**0**	**0**	**0**	168	160	176	144	**71**	**417**
RRLS *(avg.)*	8	11	8	7	5	199	202.5	–	177.5	–	–
MMAS *(avg.)*	1	3	1	1	**0**	195	184	284	164.5	219.5	851.5
THH *(best)*	1	2	**0**	1	**0**	146	173	267	169	303	1166
VNS *(best)*	**0**	**0**	**0**	**0**	**0**	317	313	375	247	292	–
FMHO *(best)*	10	9	7	17	7	243	325	249	285	132	1138
EGD *(best)*	**0**	**0**	**0**	**0**	**0**	80	105	139	88	88	730
GHH *(best)*	6	7	3	3	4	372	419	359	348	171	1068
RII *(best)*	**0**	**0**	**0**	**0**	**0**	242	161	265	181	151	–
HEA *(best)*	**0**	**0**	**0**	**0**	**0**	221	147	246	165	130	529
GD *(best)*	17	15	24	21	5	201	190	229	154	222	1066
NGD *(best)*	3	4	6	6	**0**	140	130	189	112	141	876
ENGD *(best)*	**0**	1	**0**	**0**	**0**	126	123	185	116	129	821
NGDHH-SM *(best)*	**0**	**0**	**0**	**0**	**0**	**71**	**82**	137	**55**	106	777
NGDHH-DM *(best)*	**0**	**0**	**0**	**0**	**0**	88	88	**112**	84	103	915
EMGD *(best)*	**0**	**0**	**0**	**0**	**0**	96	96	135	79	87	683

basic HSA—Proposed basic Harmony Search Algorithm.
MHSA—Proposed Modified Harmony Search Algorithm.
RRLS—Random Restart Local search (Socha et al. 2002).
MMAS—MAX-MIN Ant System (Socha et al. 2002).
THH—Tabu-search Hyper-Heuristic (Burke et al. 2003a).
VNS—Variable Neighborhood Search (Abdullah et al. 2005).
FMHO—Fuzzy Multiple Heuristic Ordering (Asmuni et al. 2005).
EGD—Extended Great Deluge (McMullan 2007).
GHH—Graph-based Hyper-Heuristic (Burke et al. 2007).
RII—Randomized Iterative Improvement (Abdullah et al. 2007b).
HEA—Hybrid Evolutionary Approach (Abdullah et al. 2007a).
GD—Great Deluge (Landa-Silva and Obit 2008).
NGD—Non-linear Great Deluge (Landa-Silva and Obit 2008).
ENGD—Evolutionary Non-linear Great Deluge (Landa-Silva and Obit 2009).
NGDHH-SM—Non-linear Great Deluge Hyper-Heuristic-Static Memory (Obit et al. 2009).
NGDHH-DM—Non-linear Great Deluge Hyper-Heuristic-Dynamic Memory (Obit et al. 2009).
EMGD—Electromagnetism Mechanism Great Deluge (Turabieh et al. 2009).

As shown in Table 6, the results obtained by basic HSA seem to be competitive with those from other previous works. These results are the best results recorded in Table 5 of the basic HSA. Note that in Table 6, the best recorded results are highlighted in bold. The basic HSA algorithm is capable of producing near optimal solutions. The results also seem to fall within the range of previous works that used the same Socha benchmarks. In addition, the MHSA is able to obtain high quality solutions for all Socha benchmarks. These results are the best recorded results from Table 4 on each Socha benchmark. Basically, the solution costs obtained by MHSA outperform the solution costs obtained by previous works in 'Medium 5' and 'Large' problem instances. The MHSA also shares the same best known results with RII, HEA, EGD, VNS, NGDHH-SM, NGDHH-DM, EMGD and some results introduced by MMSA, THH, NGD, and ENGD for small problem instances. In addition, MHSA obtains the second-best result in the 'Medium 3' problem instance. Particularly, the 'Medium 5' and 'Large' problem instances are the hardest problem instances among the Socha benchmarks as noted by Burke et al. (2007). The MHSA basically seems very effective to deal with complex and large problem instances which makes it more practical in real timetabling problems.

6 Conclusions and future work

This paper applies a harmony search algorithm to tackle the university course timetabling problem using an 11 problem instances established by Socha et al. (2002). The main rationale for developing this algorithm for timetabling stems from its potentiality to converge to the (near) optimal solution. It utilizes the advantages of population-based methods by means of recognizing the promising region in the search space using the memory consideration and randomness. It also utilizes the advantages of local search-based methods by means of fine-tuning the search space region to which it converges using the pitch adjustment operators.

We also proposed the modified harmony search algorithm (MHSA), where two modifications to the basic HSA are proposed: (i) a memory consideration is modified, and (ii) the functionality of the pitch adjustment operators is further improved by changing the acceptance rule from 'random walk' to 'first improvement' and 'side walk'.

We have deeply studied the MHSA operators by designing thirteen convergence scenarios where each of which converges to the optimal solution based on different parameter settings. We conclude that the MHSA that used the larger HMS and a larger PAR values with larger HMCR often obtains high quality solutions among all scenarios applied to Socha benchmarks.

We compare the results obtained by MHSA with the basic HSA. The results of MHSA basically outperformed those obtained by basic HSA significantly. However, the computational time needed for MHSA is longer.

The results obtained by both basic HSA and MHSA are compared to those in the literature that used the same Socha benchmarks. Generally, the basic HSA obtained results within the range of the previous works. Interestingly, the MHSA obtained high quality solutions that excel those in the previous works on two hardest Socha benchmarks. We believe that the proposed methods are highly influential with a great potential to be very valuable to the timetabling community.

It is highly recommendable that future work should be directed:

- To improve HSA for UCTP by introducing more advanced neighborhood structures in pitch adjustment procedures.
- To integrate HSA with other metaheuristic algorithms like simulated annealing acceptance rule in step 4 of HSA.
- To tune the HSA parameters for UCTP.
- To apply harmony search for different timetable forms such as examination timetabling, nurse rostering, etc.

Acknowledgements The authors would like to thank Dr. Iman Yi Liao and Dr. Munir Zaman for comments on the technical presentation. The research was partly supported by the Universiti Sains Malaysia (USM) Fellowship awarded to the first author. We would like also to thank the anonymous referees for their insightful comments.

References

Abdullah, S., Burke, E. K., & McColum, B. (2005). An investigation of variable neighbourhood search for university course timetabling. In G. Kendall, L. Lei, M. Pinedo (Eds.), *Proceedings of the 2nd multidisciplinary international conference on scheduling: theory and applications (MISTA)* (pp. 413–427). New York, USA, 18–21 July 2005.

Abdullah, S., Burke, E. K., & McCollum, B. (2007a). A hybrid evolutionary approach to the university course timetabling problem. In *CEC 2007. IEEE congress on evolutionary computation 2007* (pp. 1764–1768). Singapore.

Abdullah, S., Burke, E. K., & McCollum, B. (2007b). Using a randomised iterative improvement algorithm with composite neighbourhood structures for the university course timetabling problem. In *Metaheuristic* (pp. 153–169).

Arani, T., & Lofti, J. A. (1989). A three phased approach to final exam scheduling. *IIE Transactions*, *21*(4), 86–96.

Asmuni, H., Burke, E. K., & Garibaldi, J. M. (2005). Fuzzy multiple heuristic ordering for course timetabling. In *Proceedings of the 5th United Kingdom workshop on computational intelligence (UKCI05)* (pp. 302–309).

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys*, *35*(3), 268–308.

Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM*, *22*(4), 251–256.

Burke, E. K., & Landa-Silva, J. D. (2005). The design of memetic algorithms for scheduling and timetabling problems. In *Studies in fuzziness and soft computing*: *Vol. 166*. *Recent advances in memetic algorithms* (pp. 289–311). Berlin: Springer.

Burke, E. K., & Petrovic, S. (2002). Recent research directions in automated timetabling. *European Journal of Operational Research*, *140*(2), 266–280.

Burke, E. K., Jackson, K., Kingston, J. H., & Weare, R. (1997). Automated university timetabling: The state of the art. *The Computer Journal*, *40*(9), 565–571.

Burke, E. K., Kendall, G., & Soubeiga, E. (2003a). A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, *9*(6), 451–470.

Burke, E. K., Bykov, Y., Newall, J. P., & Petrovic, S. (2003b). A time-predefined approach to course timetabling. *Yugoslav Journal of Operations Research*, *13*, 139–151.

Burke, E. K., de Werra, D., & Kingston, J. (2004). Applications to timetabling. In J. L. Gross, & J. Yellen (Eds.), *Handbook of graph theory* (pp. 445–474). London: CRC Press.

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., & Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, *176*(1), 177–192.

Carter, M. W., & Laporte, G. (1997). Recent developments in practical course timetabling. In B. E. K., & M. C. (Eds.), *Lecture notes in computer science*: *Vol. 1408*. *The practice and theory of automated timetabling* (pp. 3–19). Berlin: Springer.

Carter, M. W., Laporte, G., & Lee, S.Y. (1996). Examination timetabling: algorithmic strategies and applications. *Journal of the Operational Research Society*, *74*, 373–383.

Chiarandini, M., Birattari, M., Socha, K., & Rossi-Doria, O. (2006). An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, *9*(5), 403–432.

Fesanghary, M., Mahdavi, M., Minary-Jolandan, M., & Alizadeh, Y. (2008). Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems. *Computer Methods in Applied Mechanics and Engineering*, *197*(33–40), 3080–3091.

Geem, Z. W. (2006). Optimal cost design of water distribution networks using harmony search. *Engineering Optimization*, *38*(3), 259–280.

Geem, Z. W. (2007a). Harmony search algorithm for solving sudoku. In B. Apolloni, R. J. Howlett, & L. Jain (Eds.), *Lecture notes in computer science (Lecture notes in artificial intelligence)*: *Vol. 4692*. *KES 2007, Part I* (pp. 371–378). Heidelberg: Springer.

Geem, Z. W. (2007b). Optimal scheduling of multiple dam system using harmony search algorithm. In F. Sandoval, A. G. Prieto, J. Cabestany, & M. Graa (Eds.), *Lecture notes in computer science*: *Vol. 4507*. *IWANN 2007* (pp. 316–323). Heidelberg: Springer.

Geem, Z. W., & Choi, J. Y. (2007). Music composition using harmony search algorithm. In M. Giacobini (Ed.), *Lecture notes in computer science*: *Vol. 4448*. *EvoWorkshops 2007* (pp. 593–600). Heidelberg: Springer.

Geem, Z. W., Kim, J. H., & Loganathan, G. V. (2001). A new heuristic optimization algorithm: harmony search. *Simulation*, *76*(2), 60–68.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading: Addison-Wesley.

Ingram, G., & Zhang, T. (2009). Overview of applications and developments in the harmony search algorithm. In Z. W. Geem (Ed.), *Music-inspired harmony search algorithm* (pp. 15–37). Berlin/Heidelberg: Springer.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings IEEE international conference on neural networks* (pp. 1942–1948).

Kostuch, P. (2005). The university course timetabling problem with a three-phase approach. In E. K. Burke, & M. A. Trick (Eds.), *Lecture notes in computer science*: *Vol. 3616*. *Practice and theory of automated timetabling* (pp. 109–125). Berlin: Springer.

Landa-Silva, D., & Obit, J. H. (2008). Great deluge with non-linear decay rate for solving course timetabling problems. In *Proceedings of the 4th international IEEE conference on intelligent systems (IS 2008)* (pp. 8.11–8.18). New York: IEEE Press.

Landa-Silva, D., & Obit, J. H. (2009). Evolutionary non-linear great deluge for university course timetabling. In E. Corchado, X. Wu, E. Oja, E. Hristozov, & T. Jedlovcnik (Eds.), *Lecture notes in computer science (Lecture notes in artificial intelligence)*: *Vol. 5572*. *Proceeding of 4th international conference on hybrid artificial intelligence systems, HAIS 2009* (pp. 269–276). Berlin/Heidelberg: Springer.

Lee, K. S., & Geem, Z. W. (2004). A new structural optimization method based on the harmony search algorithm. *Computers and Structures*, *82*(9–10), 781–798.

Lee, K. S., & 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*(36–38), 3902–3933.

Lee, K., Geem, Z. W., Lee, Sh., & Bae, Kw. (2005). The harmony search heuristic algorithm for discrete structural optimization. *Engineering Optimization*, *37*(7), 663–684.

Lewis, R. (2008). A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, *30*, 167–190.

Lewis, R., & Paechter, B. (2004). New crossover operators for timetabling with evolutionary algorithms. In A. Lofti (Ed.), *The fifth international conference on recent advances in soft computing RASC2004* (pp. 189–194). Nottingham, England.

Lewis, R., & Paechter, B. (2005). Application of the grouping genetic algorithm to university course timetabling. In G. Raidl, & J. Gottlieb (Eds.), *Evolutionary computation in combinatorial optimization (EvoCop)* (pp. 144–153). Berlin: Springer.

Lewis, R., Paechter, B., & McCollum, B. (2007). *Post enrolment based course timetabling: a description of the problem model used for track two of the second international timetabling competition*. Tech. rep., Cardiff University, Cardiff Business School, Accounting and Finance Section.

Malim, M. R., Khader, A. T., & Mustafa, A. (2006). Artificial immune algorithms for university timetabling. In E. K. Burke, H. Rudova (Eds.), *Proceedings of the 6th international conference on practice and theory of automated timetabling* (pp. 234–245). Brno, Czech Republic.

McCollum, B. (2006). University timetabling: bridging the gap between research and practice. In *Proceedings of the 5th international conference on the practice and theory of automated timetabling* (pp. 15–35). Berlin: Springer.

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A., Di Gaspero, L., Qu, R., & Burke, E. K. (2009). Setting the research agenda in automated timetabling: the second international timetabling competition. *Informs Journal on Computing*, DOI:10.1287/ijoc.1090.0320.

McMullan, P. (2007). An extended implementation of the great deluge algorithm for course timetabling. In *ICCS '07: Proceedings of the 7th international conference on computational science, Part I* (pp. 538–545). Berlin/Heidelberg: Springer.

Obit, J., Landa-Silva, D., Ouelhadj, D., & Sevaux, M. (2009). Non-linear great deluge with learning mechanism for solving the course timetabling problem. In *Proceedings of the 8th metaheuristics international conference (MIC 2009)*.

Qu, R., Burke, E. K., McCollum, B., Merlot, L. T. G., & Lee, S.Y. (2009). A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, *12*(1), 55–89.

Socha, K., Knowles, J.. & Samples, M. (2002). A max-min ant system for the university course timetabling problem. In *Springer lecture notes in computer science*: *Vol. 2463*. *Proceedings of the 3rd international workshop on ant algorithms, ANTS 2002* (pp. 1–13). Berlin: Springer.

Tuga, M., Berretta, R., & Mendes, A. (2007). A hybrid simulated annealing with kempe chain neighborhood for the university timetabling problem. In *6th IEEE/ACIS international conference on computer and information science (ICIS 2007)*, icis (pp. 400–405).

Turabieh, H., Abdullah, S., & McCollum, B. (2009). Electromagnetism-like mechanism with force decay rate great deluge for the course timetabling problem. In *Proceeding rough sets and knowledge technology (RSKT 2009)*.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transaction on Evolutionary Computation*, *1*(1), 67–82.

Yang, X. S. (2009). Harmony search as a metaheuristic algorithm. In Z. W. Geem (Ed.), *Music-inspired harmony search algorithm* (pp. 1–14). Berlin/Heidelberg: Springer.