Harmony Search as a Metaheuristic Algorithm

Xin-She Yang

Department of Engineering, University of Cambridge, Trumpington Street,
Cambridge CB2 1PZ, UK
xy227@cam.ac.uk

Abstract. This first chapter intends to review and analyze the powerful new Harmony Search (HS) algorithm in the context of metaheuristic algorithms. We will first outline the fundamental steps of HS, and show how it works. We then try to identify the characteristics of metaheuristics and analyze why HS is a good metaheuristic algorithm. We then review briefly other popular metaheuristics such as particle swarm optimization so as to find their similarities and differences with HS. Finally, we will discuss the ways to improve and develop new variants of HS, and make suggestions for further research including open questions.

Keywords: Harmony Search, Metaheuristic Algorithms, Diversification, Intensification, Optimization.

1 Introduction

When listening to a beautiful piece of classical music, who has ever wondered if there is any connection between playing music and finding an optimal solution to a tough design problem such as the water network design or other problems in engineering? Now for the first time ever, scientists have found such an interesting connection by developing a new algorithm, called Harmony Search. HS was first developed by Geem et al. in 2001 [1]. Though it is a relatively new metaheuristic algorithm, its effectiveness and advantages have been demonstrated in various applications. Since its first appearance in 2001, it has been applied to many optimization problems including function optimization, engineering optimization, design of water distribution networks, groundwater modeling, energy-saving dispatch, truss design, vehicle routing, and others [2, 3]. The possibility of combining harmony search with other algorithms such as Particle Swarm Optimization has also been investigated.

Harmony search is a music-based metaheuristic optimization algorithm. It was inspired by the observation that the aim of music is to search for a perfect state of harmony. The effort to find the harmony in music is analogous to find the optimality in an optimization process. In other words, a jazz musician's improvisation process can be compared to the search process in optimization. On one hand, the perfectly pleasing harmony is determined by the audio aesthetic standard. A musician always intends to produce a piece of music with perfect harmony. On the other hand, an optimal solution to an optimization problem should be the best solution available to the problem under the given objectives and limited by constraints. Both processes intend to produce the best or optimum.

Such similarities between two processes can be used to develop a new algorithm by learning from each other. Harmony Search is just such a successful example by transforming the qualitative improvisation process into quantitative optimization

Z.W. Geem (Ed.): Music-Inspired Harmony Search Algorithm, SCI 191, pp. 1–14.
springerlink.com

process with some idealized rules, and thus turning the beauty and harmony of music into a solution for various optimization problems.

2 Harmony Search as a Metaheuristic Method

Before we introduce the fundamentals of the HS algorithm, let us first briefly describe the way to describe the aesthetic quality of music. Then, we will discuss the pseudo code of the HS algorithm and two simple examples to demonstrate how it works.

2.1 Aesthetic Quality of Music

The aesthetic quality of a musical instrument is essentially determined by its pitch (or frequency), timbre (or sound quality), and amplitude (or loudness). Timbre is largely determined by the harmonic content that is in turn determined by the waveforms or modulations of the sound signal. However, the harmonics that it can generate will largely depend on the pitch or frequency range of the particular instrument.

Different notes have different frequencies. For example, the note A above middle C (or standard concert A4) has a fundamental frequency of f_0=*440* Hz. As the speed of sound in dry air is about v=331+0.6T m/s (where T is the temperature in degrees Celsius), the A4 note has a wavelength $\lambda=v/f_0 \approx 0.7795$ m at room temperature T=20 ℃. When we adjust the pitch, we are in fact trying to change the frequency. In music theory, pitch p_n in MIDI is often represented as a numerical scale (a linear pitch space) using the following formula

$$p_n = 69 + 12\log_2(\frac{f}{440Hz}), \tag{1}$$

or

$$f = 440 \times 2^{(p_n - 69)/12}, \tag{2}$$

which means that the A4 notes has a pitch number 69. On this scale, octaves correspond to size 12 while semitone corresponds to size 1, which leads to the fact that the ratio of frequencies of two notes that are an octave apart is 2:1. Thus, the frequency of a note is doubled (halved) when it raised (lowered) an octave. For example, A2 has a frequency of 110Hz while A5 has a frequency of 880Hz.

The measurement of harmony where different pitches occur simultaneously, like any aesthetic quality, is subjective to some extent. However, it is possible to use some standard estimation for harmony. The frequency ratio, pioneered by ancient Greek mathematician Pythagoras, is a good way for such estimation. For example, the octave with a ratio of 1:2 sounds pleasant when playing together, so are the notes with a ratio of 2:3. However, it is unlikely for any random notes played by a monkey to produce a pleasant harmony.

2.2 Harmony Search

In order to explain the Harmony Search in more detail, let us first idealize the improvisation process by a skilled musician. When a musician is improvising, he or she has three possible choices: (1) playing any famous tune exactly from his or her memory; (2) playing something similar to the aforementioned tune (thus adjusting the pitch slightly); or (3) composing new or random notes. Geem et al. formalized these three options into

quantitative optimization process in 2001, and the three corresponding components become: usage of harmony memory, pitch adjusting, and randomization [1].

The usage of harmony memory (HM) is important because it ensures that good harmonies are considered as elements of new solution vectors. In order to use this memory effectively, the HS algorithm adopts a parameter $r_{accept} \in [0,1]$, called harmony memory considering (or accepting) rate. If this rate is too low, only few elite harmonies are selected and it may converge too slowly. If this rate is extremely high (near 1), the pitches in the harmony memory are mostly used, and other ones are not explored well, leading not into good solutions. Therefore, typically, we use r_{accept}=0.7 ~ 0.95.

The second component is the pitch adjustment which has parameters such as pitch bandwidth b_{range} and pitch adjusting rate r_{pa}. As the pitch adjustment in music means changing the frequency, it means generating a slightly different value in the HS algorithm [1]. In theory, the pitch can be adjusted linearly or nonlinearly, but in practice, linear adjustment is used. So we have

$$x_{new} = x_{old} + b_{range} \times \varepsilon \tag{3}$$

where x_{old} is the existing pitch stored in the harmony memory, and x_{new} is the new pitch after the pitch adjusting action. This action produces a new pitch by adding small random amount to the existing pitch [2]. Here ε is a random number from uniform distribution with the range of [-1, 1]. Pitch adjustment is similar to the mutation operator in genetic algorithms. We can assign a pitch-adjusting rate (r_{pa}) to control the degree of the adjustment. A low pitch adjusting rate with a narrow bandwidth can slow down the convergence of HS because of the limitation in the exploration of only a small subspace of the whole search space. On the other hand, a very high pitch-adjusting rate with a wide bandwidth may cause the solution to scatter around some potential optima as in a random search. Thus, we usually use r_{pa}=0.1~0.5 in most applications.

Harmony Search

begin
  *Define objective function f(**x**), **x**=$(x_1,x_2, \ldots,x_d)^T$*
  Define harmony memory accepting rate (r_{accept})
  Define pitch adjusting rate (r_{pa}) and other parameters
  Generate Harmony Memory with random harmonies
  while *(t<max number of iterations)*
    while *(i<=number of variables)*
      if *(rand<r_{accept}), Choose a value from HM for the variable i*
        if *(rand<r_{pa}), Adjust the value by adding certain amount*
        end if
      else *Choose a random value*
      end if
    end while
    Accept the new harmony (solution) if better
  end while
  Find the current best solution
end

Fig. 1. Pseudo code of the Harmony Search algorithm

The third component is the randomization, which is to increase the diversity of the solutions. Although the pitch adjustment has a similar role, it is limited to certain area and thus corresponds to a local search. The use of randomization can drive the system further to explore various diverse solutions so as to attain the global optimality.

These three components in harmony search can be summarized as the pseudo code shown in Figure 1. In this pseudo code, we can see that the probability of randomization is

$$P_{random} = 1 - r_{accept} , \tag{4}$$

and the actual probability of the pitch adjustment is

$$P_{pitch} = r_{accept} \times r_{pa} . \tag{5}$$

2.3 Implementation

The above-mentioned three components of the HS algorithm can be easily implemented using any programming language. However, Matlab gives more straightforward way because it also provides visualization as shown in Figure 2.

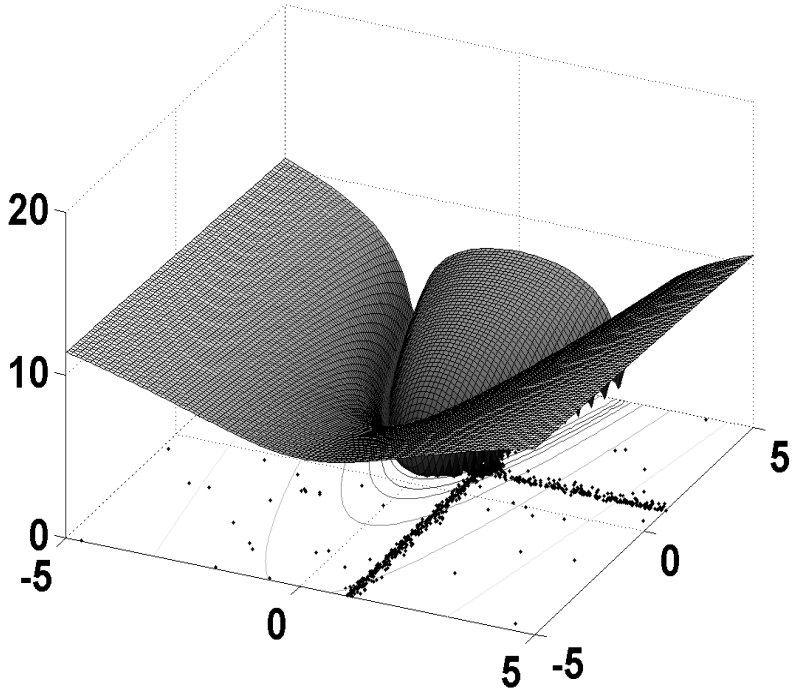

Fig. 2. The search paths of finding the global optimal solution (1,1) using the harmony search

For the first benchmark example, we test Rosenbrock's logarithmic banana function as follows:

$$f(x,y) = \ln[1+(1-x)^2+100(y-x^2)^2] \tag{6}$$

where $(x, y)\in[-10, 10]\times[-10, 10]$ and the global minimum $f_{min}=0$ at (1, 1). The HS algorithm found the global optimum successfully. The variations of these solutions and their paths are shown in Figure 2.

We have used 20 harmonies with harmony accepting rate r_{accept}=0.95, and pitch adjusting rate r_{pa}=0.7. The search paths are plotted together with the landscape of $f(x,y)$. From Figure 2, we can see that the pitch adjustment is more intensive in local regions (two thin strips).

As a further example, we present Michalewicz's bivariate function as follows:

$$f(x,y) = -\sin(x)\sin^{20}(\frac{x^2}{\pi}) - \sin(y)\sin^{20}(\frac{2y^2}{\pi}), \tag{7}$$

where a global minimum $f_{min}\approx$ -1.801 at [2.20319, 1.57049] in the domain $0\le x\le\pi$ and $0\le y\le\pi$. This global minimum was found by the HS algorithm as shown in Figure 3.

In addition to the above-mentioned two benchmark examples, this book contains many successful examples of the HS algorithm in solving various tough optimization problems, and also provides comparison among the HS algorithm and other ones. Such a comparison among different types of algorithms is still an area of active research.

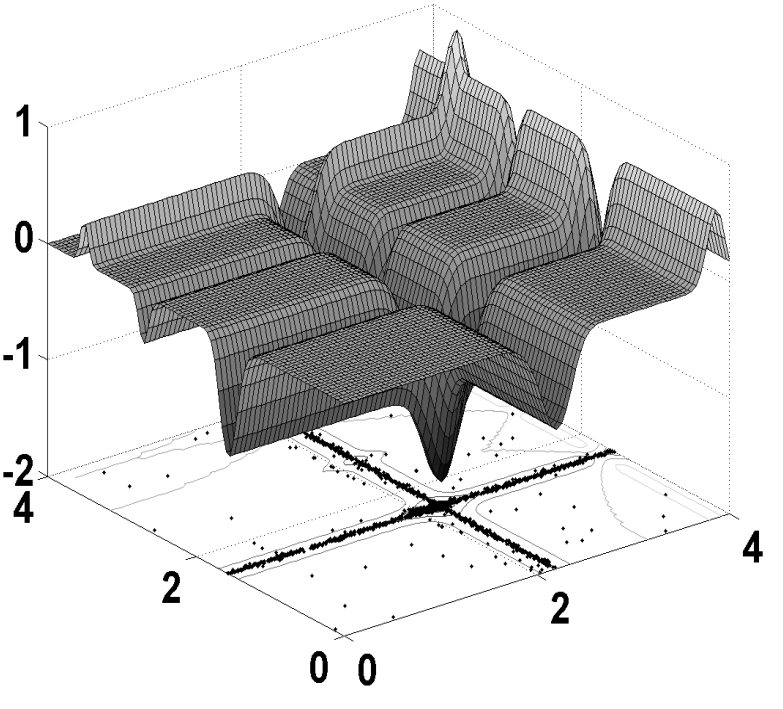

Fig. 3. Harmony search for Michalewicz's bivariate function

3 Other Metaheuristics

3.1 Metaheuristics

Heuristic algorithms typically intend to find a good solution to an optimization problem by 'trial-and-error' in a reasonable amount of computing time. Here 'heuristic' means to 'find' or 'search' by trials and errors. There is no guarantee to find the best or optimal solution, though it might find a better or improved solution than an educated guess. Broadly speaking, heuristic methods are local search methods because their searches focus on the local variations, and the optimal or best solution can locate outside of this local region. However, a high-quality feasible solution in the local region of interest is usually accepted as a good solution in many optimization problems in practice if time is the major constraint.

Metaheuristic algorithms are advanced heuristic algorithms. Because '*meta-*' means 'beyond' or 'higher-level', metaheuristic literally means to find the solution using higher-level techniques, though certain trial-and-error processes are still used. Broadly speaking, metaheuristics are considered as higher-level techniques or strategies that intend to combine lower-level techniques and tactics for exploration and exploitation of the huge solution space. In recent years, the word '*metaheuristics*' refers to all modern higher-level algorithms, including Evolutionary Algorithms (EA) including Genetic Algorithms (GA), Simulated Annealing (SA), Tabu Search (TS), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Bee Algorithms (BA), Firefly Algorithms (FA), and certainly Harmony Search [4].

There are two important components in modern metaheuristics: intensification and diversification. Such terminologies are derived from Tabu search [5]. For an algorithm to be efficient and effective, it must be able to generate a diverse range of solutions including the potentially optimal solutions so as to explore the whole search space effectively, while it intensifies its search around the neibourhood of an optimal or nearly optimal solution. In order to do so, every part of the search space must be accessible though not necessarily visited during the search. Diversification is often in the form of randomization with a random component attached to a deterministic component in order to explore the search space effectively and efficiently, while intensification is the exploitation of past solutions so as to select the potentially good solutions via elitism or use of memory or both [4-6].

Any successful metaheuristic algorithm requires a good balance between these two important, seemingly opposite, components [6]. If the intensification is too strong, only a fraction of solution space might be visited, and there is a risk of being trapped in a local optimum. It is often the case of gradient-based techniques such as Newton-Raphson method. Meanwhile, if the diversification is too strong, the algorithms converge too slowly because solutions jump around some potentially optimal solutions. Typically, the solutions start with some randomly generated (or educated guess) solutions, and gradually reduce their diversification while increasing their intensification at the same time.

Another important feature of modern metaheuristics is that an algorithm is either trajectory-based or population-based. For example, simulated annealing is a good example of trajectory-based algorithm because the path of the active search point (or agent) forms a Brownian motion-like trajectory with its movement towards some

attractors. On the other hand, genetic algorithm is a good example of population-based method since the solution search is carried out by multiple genes or agents in parallel. It is difficult to decide which type of method is more efficient as both types work almost equally successfully under appropriate conditions. There are some hints from the recent studies that population-based algorithms might be more efficient for multiobjective multimodal optimization problems as multiple search actions are in parallel. This might be true from the implementation viewpoint. However, it is far from conclusive and there exists virtually no theoretical research to back this up.

3.2 Popular Metaheuristic Algorithms

We will now briefly introduce some popular metaheuristic algorithms, and we will try to see how intensification and diversification are used.

3.2.1 Evolutionary Algorithms

Evolutionary algorithms are the name for a subset of evolutionary computation [7]. They are the search methods that were inspired from Charles Darwin's natural selection and survival of the fittest. Evolutionary algorithms are population-based search algorithms and they use genetic operators to a certain degree. These operators typically include crossover or reproductive recombination, mutation, inheritance and selection based upon their fitness.

Genetic algorithms are by far the most popular and widely used [8, 9]. However, other evolutionary search methods, including genetic programming (GP), evolutionary strategies (ES), and evolutionary programming (EP), are also popular. Briefly speaking, genetic programming is an evolutionary machine-learning technique in the framework of genetic algorithms where each individual in the population is a computer program using a scheme-style computer language such as Lisp. The objective is to find the optimal computer program to perform a user-defined task.

Evolutionary strategy is another class of nature-inspired evolutionary optimization techniques. It mainly uses mutation, selection and element-wise average for intermediate recombination as the genetic operators. It has been applied to a wide range of optimization problems.

Evolutionary programming uses arbitrary data structures and representations tailored to suit a particular problem domain, and they are combined with the essence of genetic algorithms so as to solve generalized complex optimization problems. EP is very similar to ES, but does not have the recombination operator. The main difference between EP and other methods is that EP does not use exchange of string segments, and thus there is no crossover or recombination between individuals. The primary genetic operator is mutation, often using Gaussian mutation for real-valued functions. However, its modern variants are more diverse.

Here we will briefly introduce the basic idea of genetic algorithms. Genetic algorithms were developed by John Holland in the 1960s and 1970s [8], using crossover, mutation and selection for adaptive, optimization and other problems. The essence of genetic algorithms involves the encoding of an optimization function or a set of functions into arrays of binary or character strings to represent chromosomes, and the manipulation of these strings by genetic operators with the ultimate aim to find the optimal solution. The encoding is often carried out into multiple binary strings (called

population) though real-valued encoding and representations are more often used in modern applications. The initial population then evolves by generating new-generation individuals via crossover of two randomly selected parent strings, and/or the mutation of some random bits. Whether a new individual is selected or not is based on its fitness, which is linked in some way to the objective function.

The advantages of the genetic algorithms over traditional optimization algorithms are the ability of dealing with complex problems and parallelism. GA search methods can deal with various types of optimization, whether the objective functions are stationary or non-stationary (time-dependent), linear or nonlinear, continuous or discontinuous, or even with random noise. As individuals in a population act like independent agents, each subset of the population can explore the search space in many directions simultaneously. This feature makes it ideal for the parallel implementation of the genetic algorithms.

3.2.2 Simulated Annealing

Simulated annealing is probably the best example of modern metaheuristic algorithms. It was first developed by Kirkpatrick et al. in 1983 [10], inspired by the annealing process of metals during heat treatment and also by Metropolis algorithms for Monte Carlo simulations. The basic idea of the SA algorithm is similar to dropping a bouncing ball over a landscape. As the ball bounces and looses energy, it will settle down at certain local minimum. If the ball is allowed to bounce enough time and loose energy slowly enough, the ball may eventually fall into the globally lowest location, and hence the minimum will be reached. Of course, we can use not only single ball (standard simulated annealing) but also multiple balls (parallel simulated annealing).

The optimization process typically starts from an initial guess with higher energy. It then moves to another location randomly with slightly reduced energy. The move is accepted if the new state has lower energy and the solution improves with a better objective or lower value of the objective function for minimization. However, even if the solution is not improved, it is still accepted with a probability of

$$p = \exp[-\frac{\delta E}{kT}], \tag{8}$$

which is a Boltzmann-type probability distribution. Here, T is the temperature of the system and k is the Boltzmann constant which can be set into one for simplicity. The energy difference δE is often related to the objective function $f(\boldsymbol{x})$ to be optimized. The trajectory in the simulated annealing is a piecewise path, and this is virtually a Markov chain because the new state (new solution) depends solely on current state/solution and transition probability p.

Here the diversification via randomization produces new solutions (locations). Whether the new solution is accepted or not is determined by the probability. If T is too low ($T \to 0$), then any $\delta E > 0$ (worse solution) will rarely be accepted as $p \to 0$, and the diversity of the solutions is subsequently limited. On the other hand, if T is too high, the system is at a high-energy state and most new changes will be accepted. However, the minima are not easily reached. Thus, the temperature T is an essential controlling parameter for the balance of diversification and intensification.

3.2.3 Ant Colony Optimization

Another population-based metaheuristic algorithm is the ant colony optimization (ACO) which was first formulated by Dorigo and further developed by other pioneers [11-13]. This algorithm was based upon the characteristics of behaviours of social ants. For discrete routing and scheduling problems, multiple ants are often used. Each virtual ant will preferably choose a route covered with higher pheromone concentration, and it also deposits more pheromone at the same time. If there is no previously deposited pheromone, then each ant will move randomly. In addition, the pheromone concentration will decrease gradually due to the evaporation, often with a constant evaporation rate.

Here the diversification of the solutions is represented by the randomness and the choice probability of agents along a specific route. The intensification is implicitly manipulated by the pheromone concentration and the evaporation rate. However, the evaporation rate can also affect the diversification in some way. This algorithm is exceptionally successful in the combinatorial optimization problems. Again, some fine balance between the diversification and intensification is needed to ensure a faster and efficient convergence and to ensure the quality of the solutions.

3.2.4 Particle Swarm Optimization

As genetic algorithm is population-based metaheuristics and simulated annealing is a trajectory-based one, we now introduce another population-based metaheuristic algorithm, named particle swarm optimization. The PSO was developed by Kennedy and Eberhart [14], inspired by the swarm behaviour of fish and bird schooling in nature. Unlike the single trajectory scheme used in simulated annealing, this algorithm searches the solution space by adjusting multiple trajectories of individual agents (called particles). The motion of the particles has two major components: a stochastic component and a deterministic component in terms of velocity and position vectors (solution vectors)

$$v_i^{t+1} = v_i^t + \alpha\, \varepsilon_1 (x_i - g^*) + \beta\, \varepsilon_2 (x_i - x_i^*), \quad x_i^{t+1} = x_i^t + v_i^t, \tag{9}$$

where v_i^t and x_i^t are the velocity and position of particle i at time t, respectively. ε_1 and ε_2 are two random vectors, while α and β are constants (often called the learning parameters). The diversification is controlled by the combination of random vectors and learning parameters.

The intensification is mainly represented by the deterministic motion towards the updated current best x_i^t for particle i, and the current global best g* for all particles. As the particles approach to the optima, their motion and randomness are reduced. There are many different variants of PSO in the literature [4,14].

There is a hidden or implicit feature in the PSO algorithm, that is the broadcasting ability of the current global best g^* to other particles. That can be thought as either the use of memory or some higher-level strategy so as to speed up the convergence and explore the search space more effectively and efficiently. If the diversification (or learning) parameter is large, a larger part of the search space will be explored; however, it will converge more slowly. On the other hand, high-level intensification will make the algorithm converge quickly, but not necessarily to the right solution set.

3.2.5 Firefly Algorithm

The fascinating flashing light of fireflies in the tropical summer can be used to develop interesting nature-inspired metaheuristic algorithms for optimization. The Firefly Algorithm was developed by Xin-She Yang [4], based on the idealization of the flashing characteristics of fireflies. There are three major components in the FA optimization: 1) A firefly will be attracted to more brighter or more attractive fireflies, and at the same time they will move randomly; 2) the attractiveness is proportional to the brightness of the flashing light which will decrease with distance, therefore, the attractiveness will be evaluated in the eye of the beholders (other fireflies); 3) The decrease of light intensity is controlled by the light absorption coefficient γ which is in turn linked to a characteristic scale.

The new solution is generated by

$$x_i^{t+1} = x_i^t + \alpha\, \varepsilon_1 + \beta \exp[-\gamma\, r_{ij}^2]\, \varepsilon_2 (x_i - x_j), \tag{10}$$

where r_{ij} is the distance, not necessarily the physical distance, between two fireflies i and j. In addition, FA is obviously a population-based algorithm, which may share many similarities with particle swarm optimization. In fact, it has been proved by Yang [4] that when $\gamma \to \infty$, the firefly algorithm will become an accelerated version of PSO, while $\gamma \to 0$, the FA reduces to a version of random search algorithms.

In the FA optimization, the diversification is represented by the random movement component, while the intensification is implicitly controlled by the attraction of different fireflies and the attractiveness strength β. Unlike other metaheuristics, the interaction between exploration and exploitation is intermingled in some way; this might be an important factor for its success in solving multiobjective and multimodal optimization problems.

Obviously, there are many other metaheuristic algorithms that are currently used, including, but not limited to, tabu search, cross-entropy, scatter search, cultural algorithm, flog leaping algorithm, artificial immune system, artificial bee algorithms, photosynthetic algorithm, enzyme algorithm, etc [15-20]. As we will discuss later, the hybridization of diversification and intensification components is a useful technique to develop new algorithms.

4 Characteristics of HS and Comparison

After the brief introduction to other metaheuristic algorithms, we are now ready to analyze the similarities and differences of the Harmony Search algorithm in the general context of metaheuristics.

4.1 Diversification and Intensification

In reviewing other metaheuristic algorithms, we have repetitively focused on two major components: diversification and intensification. They are also referred to as exploration and exploitation [6]. These two components are seemingly contradicting each other, but their balanced combination is crucially important to the success of any metaheuristic algorithms [4, 6].

Proper diversification or exploration makes sure that the search in solution space can explore as many locations and regions as possible in an efficient and effective manner. It also ensures that the evolving system will not be trapped in biased local optima. Diversification is often represented in the implementation as the randomization and/or additional stochastic component superposed onto the deterministic components. If the diversification is too strong, it may explore too many locations in a stochastic manner, and subsequently will slow down the convergence of the algorithm; if the diversification is too weak, there is a risk that the solution space explored is so limited and the solutions are biased and trapped in local optima, or even lead to meaningless solutions.

On the other hand, the appropriate intensification or exploitation intends to exploit the history and experience of the search process. It aims to ensure to speed up the convergence when necessary by reducing the randomness and limiting diversification. Intensification is often carried out by using memory such as in tabu search and/or elitism such as in genetic algorithm. In other algorithms, it is much more elaborate to use intensification such as the case in simulated annealing and firefly algorithms. If the intensification is too strong, it could result in premature convergence, leading to biased local optima or even meaningless solutions because the search space is not well explored. On the contrary, if the intensification is too weak, convergence becomes slow.

The optimal balance of diversification and intensification is required, and such a balance itself is an optimization process. Fine-tuning of parameters is often required to improve the efficiency of an algorithm for a particular problem. There is No Free Lunch in any optimization problem [21]. A substantial amount of studies might be required to choose the right algorithm for the right optimization problem [16], though a systematic guidance lacks for such a choice.

4.2 Why HS Is Successful

Now if we analyze the Harmony Search algorithm in the context of the major components of metaheuristics and try to compare with other metaheuristic algorithms, we can identify its ways of handling intensification and diversification, and probably understand why it is a very successful metaheuristic algorithm.

In the HS algorithm, diversification is essentially controlled by the pitch adjustment and randomization -- here there are two subcomponents for diversification, which might be an important factor for the high efficiency of the HS method. The first subcomponent of playing a new pitch (or generating a new value) via randomization would be at least at the same level of efficiency as in other algorithms that handle randomization. However, an additional subcomponent for HS diversification is the pitch adjustment operation performed with the probability of r_{pa}. Pitch adjustment is carried out by tuning the pitch within a given bandwidth. A small random amount is added to or subtracted from an existing pitch (or solution) stored in HM. Essentially, pitch adjustment is a refinement process of local solutions. Both memory consideration and pitch adjustment ensure that good local solutions are retained while the randomization makes the algorithm to explore global search space effectively. The subtlety is the fact that HS operates controlled diversification around good solutions, and intensification as well. The randomization explores the search space more widely and efficiently;

while the pitch adjustment ensures that the newly generated solution is good enough, or not too far from existing good solutions.

The intensification in the HS algorithm is represented by the harmony memory accepting rate r_{accept}. A high harmony acceptance rate means that good solutions from the history/memory are more likely to be selected or inherited. This is equivalent to a certain degree of elitism. Obviously, if the acceptance rate is too low, solutions will converge more slowly. As mentioned earlier, this intensification is enhanced by the controlled pitch adjustment. Such interactions between various components could be another important factor for the success of the HS algorithm over other algorithms, as it will be demonstrated again in other chapters of this book.

In addition, the structure of the HS algorithm is relatively easier. This advantage makes it very versatile to combine HS with other metaheuristic algorithms [22]. For algorithm parameters, there are some evidences to suggest that HS is less sensitive to chosen parameters, which means that we may not have to fine-tune these parameters to get quality solutions.

Furthermore, the HS algorithm is a population-based metaheuristic, which means that a group of multiple harmonies can be used in parallel. Proper parallelism usually leads to better performance with higher efficiency. The good combination of parallelism with elitism as well as a fine balance of intensification and diversification is the key to the success of the HS algorithm.

5 Further Research

The power and efficiency of the HS algorithm seem obvious after discussion and comparison with other metaheuristics; however, there are some unanswered questions concerning the whole class of the algorithm. Currently, the HS algorithm like other popular metaheuristics works well under appropriate conditions. However, we usually do not fully understand why and how they work so well. For example, when choosing the harmony accepting rate, we usually use a higher value, say, 0.7 to 0.95. This value is determined by experience or inspiration from genetic algorithm where the mutation rate should be low, and thus the accepting rate of the existing gene components should be high. However, it is very difficult to say what range of values and which combinations are surely better than others.

In general, there lacks a theoretical framework for metaheuristics to provide some analytical guidance to the following important issues: How to improve the efficiency for a given problem? What conditions are needed to guarantee a good rate of convergence? How to prove the global optima are reached for the given metaheuristic algorithm? These are still open questions that need further research. The encouraging thing is that many researchers are interested in tackling these difficult challenges, and important progress has been made concerning the convergence of certain algorithm (SA). Any progress concerning the convergence of HS and other algorithms would be influentially profound.

Even without a solid framework, this does not discourage scientists to develop more variants and/or hybrid algorithms. In fact, the algorithm development itself is a metaheuristic process in a similar manner to the key components of HS: using existing successful algorithms; developing slightly different variants based on the existing

algorithms; and formulating completely new metaheuristic algorithms. In using the existing algorithms, we have to identify the right algorithm for the right problem. Often, we have to change and reformulate the problem slightly and/or improve the algorithm slightly in order to find the solutions more efficiently. Sometimes, we have to develop a new algorithm from scratch to solve a tough optimization problem.

There are many ways to develop new algorithms. From the metaheuristic viewpoint, the most heuristic way is probably to develop a new algorithm by hybridization. That is to say, a new algorithm can be made based on the right combination of existing metaheuristic algorithms. For example, by combining a trajectory-type simulated annealing with multiple agents, the parallel simulated annealing can be developed. In the context of the HS algorithm, by the combination of HS with PSO, the global-best harmony search has been developed [22].

As in the case of any efficient metaheuristic algorithms, the most difficult thing is probably to find the right or optimal balance between diversity and intensity in searching the solutions. Here, the most challenging task in developing new hybrid algorithms is probably to find out the right combination of which feature/components of existing algorithms.

A further extension of the HS algorithm will be to solve multiobjective problems more naturally and more efficiently. At the moment, most of the existing studies, though very complex and tough *per se*, have mainly focused on the optimization with a single objective with or without a few criteria. The next challenges would be to use the HS algorithm to solve tough multiobjective and multicriteria NP-hard optimization problems.

Whatever the challenges will be, more HS algorithms will be applied to various problems and more systematic studies will be performed for the analysis of HS mechanism. Also, more hybrid algorithms based on HS will be developed in the future.

References

1. Geem, Z.W., Kim, J.H., Loganathan, G.V.: A new heuristic optimization algorithm: Harmony search. Simulation 76, 60–68 (2001)
2. Lee, K.S., Geem, Z.W.: A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. Comput. Methods Appl. Mech. Engrg. 194, 3902–3933 (2005)
3. Harmony Search Algorithm (2007) (accessed December 7, 2008), http://www.hydroteq.com
4. Yang, X.S.: Nature-inspired metaheuristic algorithms. Luniver Press (2008)
5. Glover, F., Laguna, M.: Tabu search. Kluwer Academic Publishers, Dordrecht (1997)
6. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput. Surv. 35, 268–308 (2003)
7. De Jong, K.: Evolutionary computation: a unified approach. MIT Press, Cambridge (2006)
8. Holland, J.H.: Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor (1975)
9. Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addison Wesley, Reading (1989)
10. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)

11. Dorigo, M., Stutzle, T.: Ant colony optimization. MIT Press, Cambridge (2004)
12. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm intelligence: from natural to artificial systems. Oxford University Press, Oxford (1999)
13. Dorigo, M., Blum, C.: Ant colony optimization theory: a survey. Theor. Comput. Sci. 344, 243–278 (2005)
14. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of IEEE Int. Conf. Neural Networks, pp. 1942–1948 (1995)
15. Yang, X.S.: Biology-derived algorithms in engineering optimization. In: Olarius, S., Zomaya, A. (eds.) Handbook of Bioinspired Algorithms and Applications. Chapman & Hall/CRC, Boca Raton (2005)
16. Yang, X.S.: Mathematical optimization: from linear programming to metaheuristics. Cambridge Int. Science Publishing, UK (2008)
17. Engelbrecht, A.P.: Fundamentals of computational swarm intelligence. Wiley, Chichester (2005)
18. Perelman, L., Ostfeld, A.: An adaptive heuristic cross-entropy algorithm for optimal design of water distribution systems. Engineering Optimization 39, 413–428 (2007)
19. Karaboga, D., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Applied Soft Computing 8, 687–697 (2008)
20. Yang, X.S.: New enzyme algorithm, Tikhonov regularization and inverse parabolic analysis. In: Simos, T., Maroulis, G. (eds.) Advances in Computational Methods in Science and Engineering – ICCMSE 2005, vol. 4, pp. 1880–1883 (2005)
21. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transaction on Evolutionary Computation 1, 67–82 (1997)
22. Omran, M., Mahdavi: Global-best harmony search. Applied Math. Computation 198, 643–656 (2008)