Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2013, Article ID 380985, 13 pages
http://dx.doi.org/10.1155/2013/380985

*Research Article*

# Generalised Adaptive Harmony Search: A Comparative Analysis of Modern Harmony Search

**Jaco Fourie,[1] Richard Green,[1] and Zong Woo Geem[2]**

[1] *Department of Computer Science and Software Engineering, University of Canterbury, Christchurch 8140, New Zealand*
[2] *Department of Energy and Information Technology, Gachon University, Seongnam 461-701, Republic of Korea*

Correspondence should be addressed to Zong Woo Geem; geem@gachon.ac.kr

Received 30 January 2013; Accepted 22 March 2013

Academic Editor: Xin-She Yang



Harmony search (HS) was introduced in 2001 as a heuristic population-based optimisation algorithm. Since then HS has become a popular alternative to other heuristic algorithms like simulated annealing and particle swarm optimisation. However, some flaws, like the need for parameter tuning, were identified and have been a topic of study for much research over the last 10 years. Many variants of HS were developed to address some of these flaws, and most of them have made substantial improvements. In this paper we compare the performance of three recent HS variants: exploratory harmony search, self-adaptive harmony search, and dynamic local-best harmony search. We compare the accuracy of these algorithms, using a set of well-known optimisation benchmark functions that include both unimodal and multimodal problems. Observations from this comparison led us to design a novel hybrid that combines the best attributes of these modern variants into a single optimiser called generalised adaptive harmony search.

## 1. Introduction

Harmony search (HS) is a relatively new *metaheuristic* optimisation algorithm first introduced in 2001 [1]. In this context, metaheuristic means that, unlike general heuristic algorithms that use trial-and-error methods to solve a specific problem, it uses higher level techniques to solve a general class of problems efficiently. HS fits into the category of population-based evolutionary algorithms together with genetic algorithms (GAs) and the particle swarm optimisation (PSO) algorithm.

As is often the case with evolutionary metaheuristics, HS was inspired by a natural phenomenon. In this case, the methods used by professional musicians to collaboratively improvise new harmonies were taken as the inspiration for HS. An analogy between improvisation and optimisation was constructed, and an algorithm was designed to mimic the way a musician uses short-term memory and past experiences to lead her to the note that results in the most pleasing harmony when played together with the other musicians. Harmony search is easy to implement and can easily be applied and adapted to solve almost any problem that can be modelled as the minimisation or maximisation of an objective function. The objective function itself can be continuous or discrete, and a smooth gradient is not required. No initial solutions or carefully chosen starting points are required. In fact, the objective function is considered a black box by harmony search. Any procedure that takes a solution vector as input and gives a fitness score as output can be used as an objective function.

These properties make harmony search attractive. It has been successfully used in a wide range of disciplines, including computer vision, vehicle routing, music composition, Sudoku puzzle, and various engineering disciplines [2–8]. However, it was soon realised that many aspects of HS can be improved and, in particular, that many of the HS parameters that are often difficult to set can be set and adjusted automatically.

An early approach to this automatic adjustment of the HS parameters was the improved harmony search (IHS) algorithm developed by Mahdavi et al. [9]. They noticed that the pitch adjustment rate (PAR) and fret width (FW) (The fret width (FW) was formally known as the bandwidth (BW), and it is still sometimes referred to as such. The terminology has since been updated to better reflect the musical analogy.) parameters are important to the fine tuning of optimised

solution vectors, and they suggested a method of adapting PAR and FW to the relative progress of the optimiser instead of keeping these parameters constant through all iterations. Their algorithm alleviates the problem of choosing an exact value for the PAR and FW, but it still requires that a range of values specified by a minimum and maximum value be given.

Another early approach to improving HS's performance and solving the parameter setting problem is the global-best harmony search (GHS) algorithm [10]. It was inspired by the swarm intelligence concept used in the particle swarm optimisation algorithm (PSO) [11]. Instead of using the FW parameter to adjust possible solutions closer to an optimum, GHS moves component values toward the value of the current best solution in the population.

Both IHS and GHS improved on the performance of HS and alleviated the parameter setting problem to some degree. However, just in the last few years, many more HS variants were developed that have shown to be superior to these early variants, and some require even less manual parameter setting. Most of these new variants are inspired by other evolutionary algorithms, and they combine the best aspects of the simplex algorithm, simulated annealing, differential evolution, sequential quadratic programming, and many other aspects with HS to form new HS variants [12–16]. For a thorough summery of the latest research in HS variants see the review article by Alia and Mandava [17].

In this paper, we concentrate on the comparison of three of the most recent and promising HS variants: exploratory harmony search (EHS), self-adaptive harmony search (SAHS), and dynamic local-best harmony search (DLHS). All three improve on both the performance of the original HS and many of the earlier HS variants. We also chose these variants because they were not developed as hybrid algorithms by combining HS with other optimisation algorithms. Instead, the focus was on investigating the steps of HS and adding novel modifications to key areas of the original algorithm.

In the section that follows, we briefly overview the HS algorithm and explain how the optimiser parameters influence the performance of the optimisation process. This provides the necessary background to introduce the three HS variants in the sections that follow. In Section 3, we compare the performance of three HS variants using a set of well-known optimisation benchmark functions. The results from these tests are then interpreted and discussed in Section 4. These observations then lead us to develop a novel hybrid algorithm called generalised adaptive harmony search (GAHS) that we introduce in Section 5. We conclude in Section 6 with final remarks and our thoughts on future research in the development of harmony search-based optimisers.

## 2. Harmony Search and Optimisers Based on Harmony Search

The harmony search algorithm is a metaheuristic population-based optimisation algorithm inspired by the way musicians in a band discover new harmonies through cooperative improvisation [1]. The analogy between the optimisation of an objective function and the improvisation of pleasing harmonies is explained by the actions of the musicians in the band. Each musician corresponds to a decision variable in the solution vector of the problem and is also a dimension in the search space. Each musician (decision variable) has a different instrument whose pitch range corresponds to a decision variable's value range. A solution vector, also called an improvisation, at a certain iteration corresponds to the musical harmony at a certain time, and the objective function corresponds to the audience's aesthetics. New improvisations are based on previously remembered good ones represented by a data structure called the harmony memory (HM).

This analogy is common to all the HS variants that we will investigate. HS variants differ in the way solution vectors (improvisations) are generated and how the HM is updated at the end of each iteration. We start our explanation of how this is done differently in each variant with an overview of the original HS algorithm.

*2.1. An Overview of Harmony Search.* The core data structure of HS is a matrix of the best solution vectors called the harmony memory (HM). The number of vectors that are simultaneously processed is known as the harmony memory size (HMS). It is one of the algorithm's parameters that has to be set manually. Memory is organised as a matrix with each row representing a solution vector and the final column representing the vector's fitness. In an $N$-dimensional problem, the HM would be represented as

$$\begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^N & | & w_1 \\ x_2^1 & x_2^2 & \cdots & x_2^N & | & w_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{\text{HMS}}^1 & x_{\text{HMS}}^2 & \cdots & x_{\text{HMS}}^N & | & w_{\text{HMS}} \end{bmatrix}, \tag{1}$$

where each row consists of $N$ decision variables and the fitness score $w$ ($[x^1, x^2, \ldots, x^N, w]$). Before optimisation starts, the HM is initialised with HMS randomly generated solution vectors. Depending on the problem, these vectors can also be randomly chosen around a seed point that may represent an area in the search space where the optimum is most likely to be found [3].

The step after initialisation is called improvisation. A new solution is improvised by using three rules: memory consideration, pitch adjustment, and random selection. Each decision variable is improvised separately, and any one of the three rules can be used for any variable. The harmony memory consideration rate (HMCR) is also one of the HS parameters that must be manually chosen. It controls how often the memory (HM) is taken into consideration during improvisation. For standard HS memory consideration means that the decisions variable's value is chosen directly from one of the solution vectors in the HM. A random number is generated for each decision variable. If it is smaller than the HMCR, the memory is taken into consideration; otherwise, a value is randomly chosen from the range of possible values for that dimension.

If the HM is taken into consideration, the improvised value is chosen randomly from one of the values in the HM. The pitch adjustment rate (PAR) is set during initialisation, and it controls the amount of *pitch adjustment* done when memory consideration is used. Another random number is generated. If it is smaller than the PAR, the improvised value is pitch adjusted using

$$x'_{\text{new}} = x_{\text{new}} + \text{rand}() \cdot \text{FW}, \tag{2}$$

where $x'_{\text{new}}$ is the new pitch-adjusted value, $x_{\text{new}}$ is the old value chosen using memory consideration, rand() is a random value between −1 and 1, and FW is the fret width parameter. The terminology has since been updated to better reflect the musical analogy [5] that controls the maximum variation in pitch adjustment and is one of the parameters that must be manually set.

Once a new value has been improvised, the memory is updated by comparing the new improvisation with the vector in the memory with the lowest fitness. If the new improvisation has a higher fitness, it replaces the vector with the lowest fitness. This process of improvisation and update continues iteratively until some stopping criterion is fulfilled, or the maximum number of iterations are reached.

To summarise, the main steps of the HS are as follows.

(1) Initialise the HM with possible solutions from the entire search space of the function to be optimised.

(2) Improvise a new solution vector using memory consideration, pitch adjustment, and random selection

(3) If the new improvisation is better than the worst solution in the HM, replace the worst solution with the new improvisation.

(4) Check the stopping criteria. If it has not been met, repeat Steps (2) and (3), or else proceed to Step (5).

(5) Return the best solution (highest fitness) in the HM as the optimal solution vector of the function.

This is a brief overview of HS. The interested reader is referred to [1, 5] for a more detailed explanation.

*2.2. Exploratory Harmony Search.* The exploratory harmony search (EHS) algorithm was the result of improving the performance of HS by focussing on the evolution of the population variance in the HM during optimisation. Das et al. realised that the exploratory power of HS can be maximised by dynamically adjusting the FW to be proportional to the standard deviation of the HM population [18].

This realisation comes from the following lemma that is proven in [18].

**Lemma 1.** *If the HMCR is chosen to be high (i.e., very near to 1), and the FW is chosen to be proportional to the standard deviation of the HM (i.e., FW* $\propto \sigma(HM) = \sqrt{\text{Var}(HM)}$*), then the expected population variance (without updating the HM) can grow exponentially over iterations.*

This lemma led them to redefine the FW as a dynamically adjusting parameter defined by $\text{FW} = k\sqrt{\text{Var}(\text{HM})}$, where $k$ is a proportionality constant. The FW is therefore recalculated at each iteration based on the current HM variance, instead of using a constant value that must be chosen manually.

To ensure exponential variance growth, the PAR, HMCR, and $k$ must also be chosen such that $((\text{HMS} - 1)/\text{HMS}) \cdot \text{HMCR} \cdot [1 + (1/3)k^2 \cdot \text{PAR}]$ is greater than 1. The proof of this statement can also be found in [18].

This simple change to HS results in an algorithm that is less likely to get stuck in local optima, requires less iterations to reach a desired accuracy, and frees the user from determining the best value for the FW parameter. However, one could argue that the FW parameter was simply replaced by the proportionality constant $k$. In their article, Das et al. investigate the effect that variations in $k$ have on the performance of EHS. They determined that EHS is not sensitive to the choice of $k$ and used a constant value ($k = 1.17$) in all their experiments. This contrasts with the choice of the FW in HS that is usually chosen to fit a particular objective function and can have a significant impact on performance if chosen poorly.

*2.3. Self-Adaptive Harmony Search.* In 2010, Wang and Huang developed a variant of HS that we refer to as the self-adaptive harmony search (SAHS) algorithm [19]. Their main focus was to alleviate the problem of choosing the best HS parameter values for a specific problem. SAHS does not require the FW and PAR parameters that need to be specified when using HS. However, SAHS like IHS requires that a minimum and maximum value for PAR be specified. Specifying a minimum and maximum for PAR is typically much easier than specifying an exact value, and it is more reliable since the performance is not as sensitive to changes in the range of PAR values as it is for a specific constant value.

SAHS is somewhat similar to IHS in that it also replaces the PAR with a range of PAR values. Like IHS, it also recalculates the current value of the PAR by linearly adjusting it between the minimum and maximum values based on the iteration count. However, it differs in that it *decreases* the PAR by starting at the maximum value and linearly decreasing it until the minimum is reached at the final iteration instead of *increasing* it like IHS does.

Unlike IHS, the FW parameter is not replaced by a range of FW values. Instead, it is replaced by a novel method of pitch adjustment that does not require an FW parameter. When pitch adjustment of an improvised decision variable is required, it is randomly adjusted for that decision variable between the minimum and maximum values found in the current HM, using the following equations:

$$\begin{aligned} x'_i &\longleftarrow x'_i + \left[\max\left(\text{HM}^i\right) - x'_i\right] \cdot r, \\ x'_i &\longleftarrow x'_i - \left[x'_i - \min\left(\text{HM}^i\right)\right] \cdot r, \end{aligned} \tag{3}$$

where $x'_i$ is the new improvisation, $\max(\text{HM}^i)$ and $\min(\text{HM}^i)$ are the maximum and minimum values in the HM for the $i$th decision variables and $r$ is a uniformly generated random number between 0 and 1. Each time pitch adjustment is performed, the improvisation is updated by randomly applying, with equal probability, one of these two equations.

This approach causes progressively smaller changes to be made to the new improvisation as $\max(\mathrm{HM}^i)$ and $\min(\mathrm{HM}^i)$ converge closer together with increasing iterations. Therefore, pitch adjustment begins as a rough operator making larger changes to favour exploration and then becomes a fine operator favouring exploitation as the optimiser converges closer to the optimum.

Another important difference between SAHS and HS is in the initialisation of the HM. In HS, the HM is initialised one decision variable at a time by randomly picking a value from a uniform distribution of values in the allowable range. However, this uniform distribution is only an approximation created by scaling the output from a pseudorandom number generator to the correct range of values. The resulting distribution is often unevenly distributed causing slow convergence or convergence to local optima. Instead of using a pseudorandom number generator, SAHS uses a low discrepancy sequence [20] to initialise the HM. Low discrepancy sequences are more evenly spread out in the search space than pseudorandom ones, causing the initial vectors in the HM to better represent the search space of the problem.

*2.4. Dynamic Local-Best Harmony Search.* dynamic local-best harmony search (DLHS) was developed by Pan et al. as an improvement to IHS and GHS that does not require the HMCR and PAR parameters [21]. The focus of DLHS can be roughly divided into three categories. First, the improvisation step was improved by basing new improvisations on the current best solution vector in the HM instead of a randomly chosen one. Second, it divides the HM into multiple independently optimised sub-HMs in an attempt to maintain greater diversity throughout the search process. Third, DLHS uses a self-learning parameter set list (PSL) to evolve the HMCR and PAR parameters into the optimal values without any user specification. The FW parameter is dynamically adjusted in a similar way to the method that IHS uses.

The modification of the improvisation process is inspired by GHS. Like GHS, the random solution vector used in memory consideration is replaced by the best solution vector in the HM. This tends to concentrate future improvisations around the current best solution which is likely a local optimum, and it may be the global optimum. To prevent possible convergence around local optima, the random solution vector that was discarded in memory consideration is instead used during pitch adjustment. So unlike the pitch adjustment of (2), DLHS uses the following equation:

$$x'_{\mathrm{new}} = x_{\mathrm{rand}} + \mathrm{rand}\,() \cdot \mathrm{FW}, \tag{4}$$

where $x_{\mathrm{rand}}$ is randomly chosen from the current values in the HM for a particular decision variable.

Before improvisation starts, the HM is randomly divided into $m$ equally sized sub-HMs. Each sub-HM then uses its own small subset of solution vectors to independently converge onto a local (or possibly global) optimum. Due to the loss of diversity that comes with restricting the size of the individual HMs by splitting them up into sub-HMs, it is likely that convergence will not be towards the global optimum but rather toward a local one. To prevent this loss of diversity, information is allowed to exchange between sub-HMs with a frequency controlled by the regrouping schedule $R$. The regrouping schedule determines how often information exchange is allowed between sub-HMs by randomly regrouping the individual solution vectors of all sub-HMs into $m$ new sub-HM configurations every $R$ iteration. This regrouping simultaneously increases the diversity of all sub-HMs and allows the best solutions from the entire HM to be shared among sub-HMs.

Once the global optimum is identified, the injection of diversity through the regrouping operation does not further optimise, but it can result in inaccurate convergence. For this reason, DLHS enters the final phase, of the optimisation process after 90% of the iterations have been made. In this final phase the regrouping operation is halted, and a new HM is formed by combining the best three solution vectors from all sub-HMs into a single new HM. The new HM is then exclusively processed until the maximum number of iterations has been reached.

The HMCR and PAR parameter values are dynamically determined by selecting from a self-learning parameter set list (PSL). This process starts with the initialisation of the PSL by filling it with randomly generated HMCR and PAR values. HMCR values are generated from a uniform distribution where $\mathrm{HMCR} \in [0.9, 1.0]$ and PAR values are generated from a uniform distribution where $\mathrm{PAR} \in [0.0, 1.0]$. At the start of each iteration, one pair of HMCR and PAR values is removed from the PSL and used for that iteration. If the current iteration's improvisation resulted in the HM being updated, the current parameter pair is saved in the winning parameter set list (WPSL). This process continues until the PSL becomes empty. The PSL is then refilled by randomly selecting a pair from the WPSL 75% of the time and randomly generating a new pair 25% of the time. To prevent memory effects in the WPSL, the WPSL is emptied each time the PSL is refilled. The result of this is that the best parameter set is gradually learned, and it is specific to the objective function under consideration. The size of the PSL is set to 200 pairs. It was found that the effect that the size of the PSL has on performance is insignificant [21].

Unlike the HMCR and the PAR, the FW parameter is dynamically adjusted in a way that is similar to the way IHS adjusts the PAR. Like IHS, DLHS favours a large FW during the early iterations to encourage exploration of the search space, while a small FW is favoured during the final iterations to better exploit good solutions in the HM. The FW is therefore linearly decreased with increasing iterations using the following equation:

$$\mathrm{FW}(i) = \begin{cases} \mathrm{FW}_{\max} - \dfrac{\mathrm{FW}_{\max} - \mathrm{FW}_{\min}}{\mathrm{MI}} 2i & \text{if } i < \dfrac{\mathrm{MI}}{2} \\ \mathrm{FW}_{\min} & \text{otherwise,} \end{cases} \tag{5}$$

where $\mathrm{FW}_{\max}$ and $\mathrm{FW}_{\min}$ are the maximum and minimum values of the FW, $i$ is the iteration number, and MI is the maximum number of iterations. Like IHS, this means that a minimum and maximum value for the FW has to be set before optimisation starts, but this is again assumed to be much easier than deciding on a single value for the FW.

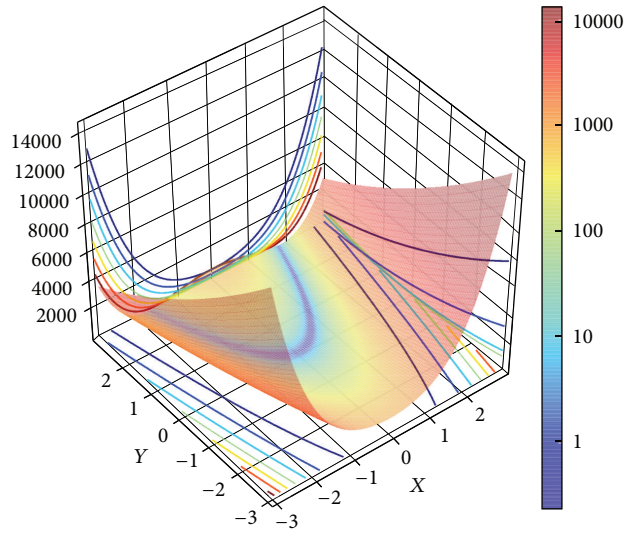


Figure 1: Rosenbrock's valley is illustrated here in two dimensions. Notice the parabolic valley that includes the global optimum at $[1, 1]$.

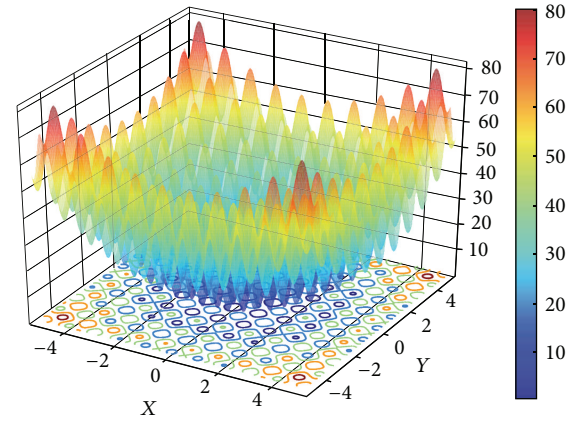


Figure 2: This illustration of Rastrigin's function shows its highly multimodal nature, and it is difficult to visually identify the global optima at $[0, 0]$.

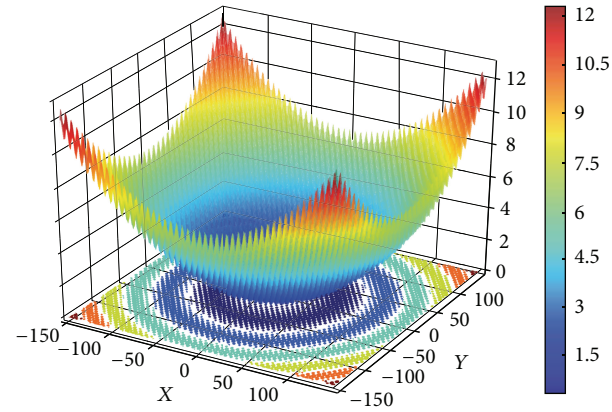


Figure 3: This is a two-dimensional surface plot of Griewangk's function. The global optima are at $[0, 0]$, where the function evaluates to 0.

## 3. Performance Analysis Using Five Benchmark Functions

*3.1. Benchmark Functions.* We start our performance analysis with a brief description of the five benchmark functions that we will be using to compare performance. All five of these functions were designed to be challenging to optimise for different reasons, and all have previously been used for benchmarking harmony search-based algorithms.

We start with a classic parametric optimisation function called Rosenbrock's valley [22]. This multidimensional problem has a global optimum inside a long narrow parabolic-shaped flat valley. The valley itself is easy to find since there are no other local minima anywhere in the search space, but converging onto the global optimum is difficult. Rosenbrock's valley is defined by the following equation:

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} 100\left(x_{i+1} - x_i^2\right) + \left(1 - x_i\right)^2 \qquad -2.048 \le x_i \le 2.048. \tag{6}$$

Rosenbrock's valley has a global optimum at $x_i = 1\ \forall i$, where the function evaluates to 0. An illustration of Rosenbrock's valley implemented in two dimensions is seen in Figure 1.

Rastrigin's function uses cosine modulation to create a highly multimodal function that often causes optimisation algorithms to get stuck in the local optima without ever finding the global optimum [23]. However, it does have a single global optimum at $x_i = 0\ \forall i$, where the function evaluates to 0. Rastrigin's function is defined by

$$f(\mathbf{x}) = 10n + \sum_{i=1}^{n}\left(x_i^2 - 10\cos\left(2\pi x_i\right)\right) \qquad -5.12 \le x_i \le 5.12. \tag{7}$$

An illustration of this function in two dimensions is seen in Figure 2. One can clearly see the multiple local minima in the contours overlaid on the floor of the surface plot.

Griewangk's function is similar to Rastrigin's function in that it also uses cosine modulation to create a highly multimodal function [23]. However, Griewangk's function is optimised over a larger search space and has different properties as one gets closer to the global optima situated at $x_i = 0\ \forall i$.

When one considers the entire search space, the function looks like a simple multidimensional unimodal parabola which could easily be minimised using gradient descent. As one moves closer to the optimum and starts to consider smaller areas, it becomes clear that the function is not as smooth as first thought, and it is in fact highly multimodal. This is clearly illustrated in Figure 3 where a surface plot of a two-dimensional implementation of Griewangk's function

is shown. The multimodal nature of Griewangk's function is also apparent from its defining equation which is as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \qquad (8)$$
$$-600 \leq x_i \leq 600.$$

Ackley's Path function is another multimodal test function that is widely used to benchmark optimisation algorithms [24]. It uses a combination of exponentials and cosine modulation to create a search space with many local optima and a single global optimum at $x_i = 0\ \forall i$, where the function evaluates to 0. Ackley's Path function is illustrated in Figure 4 and is defined by the following equation:

$$f(\mathbf{x}) = 20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e \qquad (9)$$
$$-32.768 \leq x_i \leq 32.768.$$

The last test function that we consider is the Goldstein-Price function. This eighth-order polynomial is defined in two variables only and has global minima at $[0.0, -1.0]$ where the function evaluates to 3.0. This function has four local minima that lie close together, and the search space is usually limited to the $-2 \leq x_1, x_2 \leq 2$ cube. The Goldstein-Price function is illustrated in Figure 5 and is defined by the following polynomial:

$$f(\mathbf{x}) = \left(1 + (x_1 + x_2 + 1)^2 \times \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right) \times \left(30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right)$$
$$-32.768 \leq x_i \leq 32.768. \qquad (10)$$

*3.2. Test Results.* We compared the performance of the three HS variants together with the original HS algorithm using a series of test runs over all five benchmark functions. We implemented all the benchmark functions except for the Goldstein-Price function in both 30 and 100 dimensions to measure performance in both low- and high-dimensional problems. In functions with 100 dimensions, the algorithms were allowed to optimise for 500,000 function evaluations, while those in 30 dimensions were allowed to run for 100,000. The Goldstein-Price polynomial is only defined in two dimensions and therefore requires considerably less iterations to optimise. We allowed only 10,000 function evaluations when the Goldstein-Price polynomial was optimised.

In our first experiment, all HM variants were implemented using the parameters suggested by the original

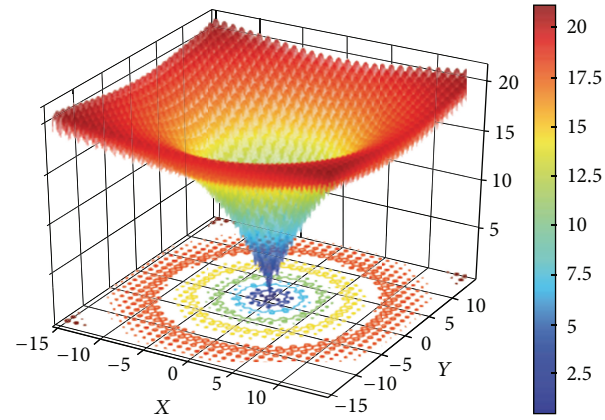


Figure 4: This is a two-dimensional surface plot of Ackley's function. The global optima is at $[0, 0]$ where the function evaluates to 0.

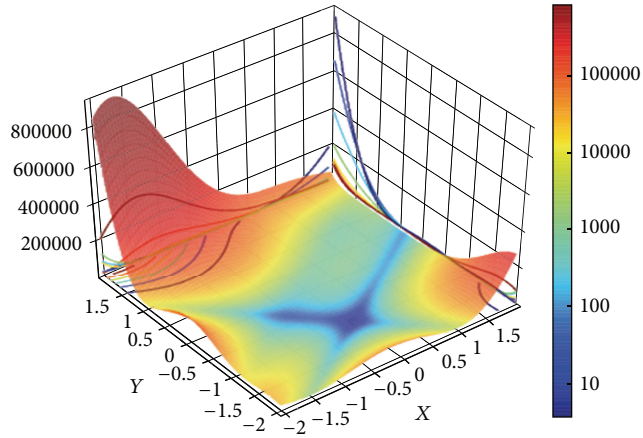


Figure 5: The Goldstein-Price polynomial is only defined in two variables and is shown here evaluated in the $-2 \leq x_1, x_2 \leq 2$ cube.

authors. The only exceptions were the parameter sets used in the original harmony search algorithm and that used for DLHS. Many different *optimal* parameter sets have been suggested for HS by researchers, and we chose to find our own set based on the best overall results after simulating all of five benchmark functions. For DLHS, the authors suggest 3 sub-HMs of size 3 which is much smaller than the 50 element HMs suggested by all the other HM variants [21]. We therefore opted to use 5 sub-HMs of size 10 instead, both because this resulted in better results for DLHS and also for a more fair comparison with the other variants. The parameter sets that we used in all our experiments are indicated in Table 1.

Each of the five benchmark functions were minimised using these parameters. The results for the 30 dimensional problems (together with the Goldstein-Price problem) are shown in Table 2. In each experiment, three values are given. The first is the average score calculated over 50 independently initialised runs. This is followed by the standard deviation and

Table 1: Parameter sets as suggested by the original authors.

| | HS | EHS | DLHS | AHS |
|---|---|---|---|---|
| HMS | 50.00 | 50.00 | 50.00 | 50.00 |
| HMCR | 0.99 | 0.99 | — | 0.99 |
| PAR | 0.33 | 0.33 | — | — |
| $PAR_{min}$ | — | — | — | 0.00 |
| $PAR_{max}$ | — | — | — | 1.00 |
| FW | 0.01 | — | — | — |
| $FW_{min}$ | — | — | 0.0001 | — |
| $FW_{max}$ | — | — | (UB − LB)/200† | — |
| $k$ (EHS) | — | 1.17 | — | — |
| $R$ (DLHS) | — | — | 50.00 | — |
| $m$ (DLHS) | | | 5.00 | |

†UB and LB refer to the lower and upper bound of the component values.

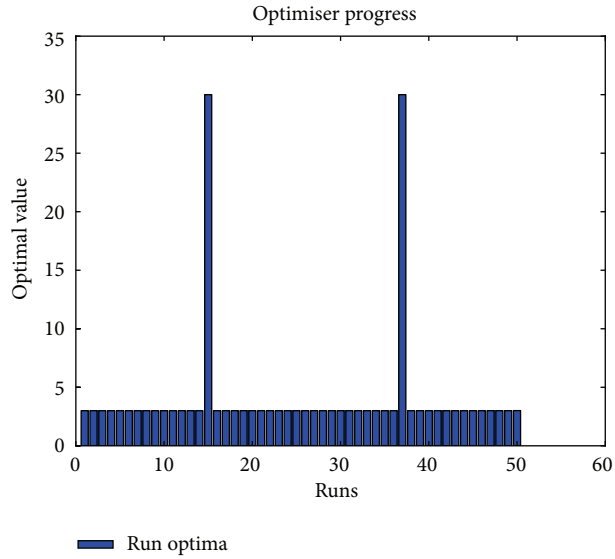


Figure 6: This bar graph shows the result of 50 independent runs of EHS on the Goldstein-Price polynomial. Notice that 48 of the 50 runs converged to the correct global minimum, while two converged to one of the local minima that is 10 times larger than the global minimum.

then the success rate defined as the number of successful hits on the global minimum within a 0.01 tolerance.

Notice that in many cases, more than one algorithm achieved 100% success rate (shown as 50 successful hits) on the global minimum. In these cases, we define the best results as the algorithm with both a 100% success rate and an average score closest to the global minimum. We repeat this experiment using the same parameter values on 100-dimensional problems. The average scores from the second experiment are shown in Table 3.

Some of these values seem surprising, and it is worth investigating why this is so. One would expect that the 2-dimensional Goldstein-Price function would be the easiest to minimise and that all the average scores should be very close to 3.0 which is the global optimum. However, as seen in Table 2, this is not the case for HS, DLHS, and EHS. These algorithms are not necessarily so much worse than AHS, but this indicates that some of the 50 runs that contributed to the average score converged to a local minimum that was far different from the global one. Since these local minima may be much larger than the global minimum, it can have a large effect on the average score. We see this clearly happening in the graph of Figure 6. The graph shows that only 2 of the 50 runs failed to converge to the exact global minimum, but because of these 2 the average score is significantly different than the global minimum.

A quick comparison of the results presented in Tables 2 and 3 suggests that the AHS algorithm is the overall best optimiser. In the 30-dimensional experiment, it performed the best on two of the five benchmark functions and was the only optimiser to achieve a 100% success rate on the Goldstein-Price polynomial. Its performance in optimising Ackley's, Griewank's and Rosenbrock's functions is very similar to that of EHS, and there are no significant differences that might indicate that one is clearly better than the other when only these three functions are considered. However, both completely failed at minimising Rosenbrock's function. It is intersting that the only successful run from all algorithms when optimising Rosenbrock's function was achieved by the original unmodified HS algorithm. This shows that the original HS algorithm can perform equally well and sometimes even better than modern variants when an optimised parameter set is used.

Surprising results are also seen in Rastrigin's column of Table 2. Like with Rosenbrock function, both EHS and AHS failed to optimise this function, but unlike Rosenbrock's the DLHS optimiser performs well here and has a much better hit ratio than any of the other variants. In the section that follows, we suggest a possible explanation for this result and investigate how one can use this to design an improved HS variant.

The results from the 100-dimensional problem summarised in Table 3 are very similar to that of the 30-dimensional problem. For Ackley's, Griewank's and Rosenbrock's functions, the performance of EHS and AHS is again the best and is not significantly different. Like in the 30-dimensional problem, both EHS and AHS failed in optimising Rastrigin's function which is again optimised much more successfully by DLHS.

## 4. Interpretation of Results

In this section, results from Tables 2 and 3 are used to show the best attributes of the HS variants and where they fail. In the previous section, we saw that EHS and AHS generally perform the best, but both failed to optimise Rastrigin's function. DLHS performs much better on Rastrigin's function but tends to get stuck in local optima as seen in the results from 30-dimensional Ackley's and the Goldstein-Price functions. The question is then why does DLHS perform so much better

Table 2: Average scores and standard deviations over 50 runs for 30-dimensional problems. All optimisers were allowed to run for 100,000 function evaluations, except when the Goldstein-Price polynomial was being minimised. In that case, 10,000 evaluations were allowed. The best results from each benchmark function are indicated in bold.

| | Ackley | Griewank | Rosenbrock | Rastrigin | Goldstein-Price[†] |
|---|---|---|---|---|---|
| HS | | | | | |
| Best | 0.0066 | 0.0322 | **30.7287** | 0.0142 | 6.8464 |
| Standard deviation | 0.0006 | 0.0334 | **22.6459** | 0.0027 | 9.3353 |
| Hits | 50 | 48 | **1** | 2 | 42 |
| DLHS | | | | | |
| Best | 0.8464 | 0.0170 | 32.7372 | **0.2003** | 6.2444 |
| Standard deviation | 0.5173 | 0.0180 | 22.8117 | **0.4200** | 8.7724 |
| Hits | 2 | 50 | 0 | **31** | 44 |
| EHS | | | | | |
| Best | 0.0008 | **0.0006** | 28.9183 | 15.4119 | 4.0809 |
| Standard deviation | 0.0024 | **0.0022** | 10.7493 | 10.8267 | 5.2910 |
| Hits | 50 | **50** | 0 | 5 | 48 |
| AHS | | | | | |
| Best | **0.0003** | 0.0042 | 26.5041 | 1.4819 | **3** |
| Standard deviation | **0.0020** | 0.0111 | 0.5624 | 0.9055 | **0** |
| Hits | **50** | 50 | 0 | 1 | **50** |

[†]Goldstein-Price was implemented in two dimensions as it is only defined in two.

Table 3: Average scores and standard deviations over 50 runs for 100-dimensional problems. The optimisers were allowed to run for 500,000 function evaluations.

| | Ackley | Griewank | Rosenbrock | Rastrigin[†] |
|---|---|---|---|---|
| HS | | | | |
| Best | 0.0142 | 0.0050 | 162.5829 | 0.2280 |
| Standard deviation | 0.0004 | 0.0080 | 49.5829 | 0.0130 |
| Hits | 50 | 34 | 0 | 0 |
| DLHS | | | | |
| Best | 0.2281 | 0.0004 | 142.4515 | **1.3140** |
| Standard deviation | 0.0130 | 0.0070 | 39.5126 | **3.1364** |
| Hits | 50 | 50 | 0 | **32** |
| EHS | | | | |
| Best | 0.0009 | 0.0001 | **97.2555** | 388.2894 |
| Standard deviation | 0.0006 | 0.0007 | **7.4711** | 20.5949 |
| Hits | 50 | 50 | **0** | 0 |
| AHS | | | | |
| Best | **0.0003** | **0** | 96.4333 | 6.7471 |
| Standard deviation | **0.0006** | **0** | 0.2943 | 2.3279 |
| Hits | **50** | **50** | 0 | 0 |

[†]The accuracy tolerance for awarding a successful hit on the global minimum was decreased to 0.1 for 100-dimensional Rastrigin's function.

than EHS and AHS on Rastrigin when it is worse in all other examples?

We start to answer that question by investigating what novel improvement these HS variants contribute to the final result. The main contribution of both EHS and AHS is the dynamic adjustment of the aggressiveness of the pitch adjustment operator. EHS does this by dynamically adjusting the value of the FW based on the variance of the values currently in the HM. AHS also determines the amount of pitch adjustment by analysing the values in the HM, but it does so without needing to directly calculate the variance. In both cases the effect is that pitch adjustment is more

aggressive at the beginning of the optimisation process and continues to make smaller and smaller adjustments as the variance in the HM decreases due to convergence.

The advantage that each of these approaches brings is that the pitch adjustment operator dynamically shifts from an aggressive mode that favours exploration of the search space at the start of optimisation to a less aggressive mode that favours exploitation of possible minima that were discovered during the aggressive mode. However, this also means that when the optimiser converges to local minima, the drop in variance of values in the HM causes the pitch adjustment to favour exploitation, making it unlikely that the optimiser will escape the local minima and find the global one. Premature convergence to local optima is therefore a weakness of these methods and may explain why even the original HS scored better than EHS and AHS on the optimisation of Rastrigin's function.

AHS attempts to minimise the effect of this weakness using the adjustment of the PAR. By linearly decreasing the PAR from a very high to a very low value, the exploration phase is lengthened due to the aggressive and frequent pitch adjustment caused by a high PAR and a high variance in the HM at the start of optimisation. In practice this does have a positive effect on the results and may explain the advantage AHS had over EHS in the Goldstein-Price experiment. It was, however, still not enough to prevent the poor performance when optimising Rastrigin's function.

DLHS uses a completely different approach, first maximising the HM diversity by dividing it up into sub-HMs and then by dynamically adjusting the PAR, FW, and HMCR values to fit the function being optimised and the optimisation progress. The sub-HM idea has been used successfully before to maximise diversity in harmony search (see [4, 25]). This idea of separating the population of candidate solutions into groups that converge independently is also used successfully in other evolutionary algorithms, and it was originally used in genetic algorithms in what became known as island model parallel genetic algorithms [26, 27].

The weakness of this approach, however, is often slow convergence due to function evaluations that are wasted in subpopulations (sub-HMs) and end up converging to local optima or not converging at all. Since the subpopulations converge independently, it is likely that they will converge to different optima. This is actually the desired behaviour since the motivation of this approach was the maximisation of diversity in the candidate solutions. However, the question then becomes when to consolidate the independent results from each subpopulation so the best candidates can be exploited for more accurate convergence.

The answer to this question lies in the classic tradeoff between maximising the convergence speed and minimising the probability of premature convergence to local optima. If sub-populations converge independently for too long, iterations are wasted on results that will finally be discarded. On the other hand, when results are consolidated too quickly diversity among the sub-populations is lost, and this increases the risk of premature convergence.

In DLHS, this tradeoff is left as a parameter, namely the regrouping schedule, and it is up to the user to decide how aggressively the sub-HMs should be consolidated. It is difficult to choose the best value for this parameter, and the optimum value is likely problem specific. This means that a single good choice is unlikely. The original authors of DLHS suggested a value of 50 iterations for the regrouping schedule. Rastrigin's function was one of the functions they used to test their algorithm [21]. This may explain the good results that DLHS produced on Rastrigin's function.

Another major contributing factor to the DLHS results is the dynamic adjustment of the PAR, FW and HMCR parameters. Like EHS and AHS, DLHS starts with a large FW, and decreases it as the optimiser progresses. However, unlike EHS and AHS, it does not take the variance in the current HM into account but instead linearly decreases it proportionally to the iteration count. This is similar to the way the IHS algorithm operates (see [9]) and to the way AHS linearly decreases the PAR. We believe that not taking the variance in the HM into account is a weakness of DLHS, especially since the consolidation of the sub-HMs can have a dramatic effect on the HM causing changes in the variance that do not follow a linearly decreasing trend.

However, the PAR and HMCR are adjusted in a novel way that potentially has the largest impact on the quality of the final results. To our knowledge, DLHS is also the only HM variant to dynamically adjust the HMCR. The HMCR is a parameter that has previously been shown by several researchers, including the authors of AHS and EHS, to have a large effect on the quality of the final results [10, 18, 19].

In our own analysis (see Section 2.4) of the values that end in the WPSL after several thousand iterations, we noticed that the self-adapting process that controls the PAR and HMCR values tends to favour very small PARs ($<0.01$) and large HMCRs ($>0.99$). This seems to verify our choice of optimal values for the HMCR and the PAR, but it also suggests that the maximum PAR used in AHS should optimally be much smaller than 1.0.

Finally, one should also be very aware of the fact that because the HM is divided into sub-HMs, the actual HM improvisation steps are performed using a much smaller HMS than the 50 candidates that were chosen for the other HM variants. Like the HMCR, HMS also makes a large difference to the quality of the results, but it is one of the hardest parameters to choose [10, 19]. Researchers initially recommended a small HMS ($<15$), but the authors of AHS showed that AHS performs better with a larger HMS which was also the conclusion made by the authors of EHS. However, in our own experiments, we found that AHS performs much better on Rastrigin's function, even to the point of being better than DLHS, when the HMS is decreased to 10 instead of the suggested 50. However, this caused a decrease in performance when optimising Griewank's and Ackley's functions. This then suggests that the HMS should be chosen to fit the problem, or it should be dynamically adjusted like the PAR and FW.

It is currently not known how the HMS should be adjusted or modelled to a specific problem. The practical effects that this would have on the HM would be much greater than simply changing the HMCR or the PAR since candidates will need to be added or removed from the HM. A further

problem would then be determining which candidate should be removed when the HMS is decreased, and how should a new candidate be created when the HMS is increased. If we are to develop an HM variant that performs well on all our benchmark functions without requiring a fine-tuned parameter set unique to each problem, this is the type of question that will need to be answered.

## 5. Generalised Adaptive Harmony Search

Our approach to designing an HS variant that performs well on all our benchmark functions is based on the AHS algorithm. We call it the generalised adaptive harmony search (GAHS) algorithm. We chose to base GAHS on AHS because of its simplicity and its good performance on all but Rastrigin's function. As we have pointed out, AHS can be made to perform well on Rastrigin given the right parameter set, but no single parameter set seems to perform well on all the benchmark functions.

We therefore need to either find a way to dynamically adjust the parameters during optimisation to fit the particular problem (particularly the HMS) or augment the improvisation process some other way that would allow for the same benefits. Our first attempt at addressing this was to split the HM into sub-HMs in the same way that DLHS does. It was our hope that this would both decrease the effect of HMS, which we already know has a positive effect on the results for Rastrigin's function and increase diversity in the HM enough to compensate for AHS's tendency for premature convergence. We used the AHS pitch adjustment operator and linearly decreased the PAR with the intention that this would compensate for DLHS's slow convergence.

Our initial results showed that using sub-HMs in AHS improved the performance when minimising Rastrigin's function but the convergence rate also slowed considerably, resulting in poor performance when minimising Griewank and Ackley. Several parameter sets were tried which indicated that a longer regrouping period was needed to compensate for AHS's tendency for premature convergence. Larger sub-HMs resulted in good performance in Griewank and Ackley, while Rastrigin needed small sub-HMs like those DLHS uses. However, no single parameter set that performed well on all our benchmark functions could be found.

A better approach was needed, and it was clear that using sub-HMs will either result in slow convergence or the need to fine-tune a parameter set to a specific problem. The focus should be on the HMS parameter as this parameter has the greatest effect on the performance of AHS when Rastrigin's function is minimised.

We propose to find a small collection of parameter sets, in particular HMS and HMCR values, that includes at least one set that will result in good performance on any of our benchmark functions. An instance of AHS will then be started for each parameter set and run for a fraction of the total function evaluations allowed. Once this has been done for each parameter set, the best results from each HM is compared, and the instance with the worst result is dropped. The process then repeats until only the best instance with the most optimal parameter set remains. This instance is then left to converge until the remainder of the allowable function evaluations have been reached. All AHS instances are initialised using a low discrepancy sequence (see Section 2.3), and regroupings of candidates between separate instances are never done. Each instance therefore converges independently.

This approach requires the addition of two important parameters. The first, called the period length (PL), is the number of iterations that a particular instance is allowed to run before it is stopped and evaluated to determine whether more function evaluations are spent on it or whether it is abandoned. The second is the *parameter set collection size* (PSCS) which is the number of parameter sets in the collection and is therefore also the number of AHS instances that are compared. These two parameters are important because together they determine the percentage of the total function evaluations wasted on finding the optimal parameter set and how many are left for convergence to the global optimum. Since one function evaluation is done during each iteration of AHS, the total number of iterations spent on instances that do not contribute to the final result (wasted iterations) can be calculated using the following equation:

$$\text{Wasted iterations} = \text{PL} \sum_{i=2}^{\text{PSCS}} (i-1)\,. \tag{11}$$

To maximise the number of useful iterations, the PL and the PSCS is kept as low as possible. However, if the PSCS are too small, an important parameter set may be overlooked. If the PL is too small, some of the AHS instances may be discarded before an accurate estimate of their quality can be measured.

Through experimentation, we determined that an optimal parameter set collection contains three parameter sets. All three parameter sets are equal to that suggested by the original authors of AHS and are identical except for the HMS. We use three different values for the HMS, namely, 10, 30, and 50. A good value for the PL was found to be 10% of the total number of iterations. This means according to (11) that 30% of the total iterations is spent on finding the optimal parameter set and 70% for convergence to the global optimum.

The performance of GAHS as defined in the previous paragraphs is compared with the other HS variants in Table 4. The benchmark functions were again implemented in 100 dimensions. We repeat the results from Table 3 for comparison.

It is seen from this comparison that GAHS performs even better than DLHS on Rastrigin's function as well as AHS on the other benchmark functions. It still performs poorly on Rosenbrock's function. Even when we used a fine-tuned parameter set specific to Rosenbrock's function, we could not improve the results. We suspect that since EHS also converges to this point that large local optima exist in that area, and that the nature of this function is such that escaping from this local optimum is very difficult for harmony search-based optimisers.

It is also clear that some weaknesses that hinder GAHS still remain. Since GAHS is only a generalised form of AHS,

Table 4: In this table, the performance of GAHS is compared with the other HS variants. All scores are averaged over 50 runs, and all functions except Goldstein-Price are implemented in 100 dimensions. The optimisers were allowed to run for 500,000 function evaluations except for Goldstein-Price which was allowed only 10,000 evaluations. The best results are again indicated in bold.

| | Ackley | Griewank | Rosenbrock | Rastrigin[†] | Goldstein-Price[%] |
|---|---|---|---|---|---|
| HS | | | | | |
| Best | 0.0142 | 0.0050 | 162.5829 | 0.2280 | 6.8464 |
| Standard deviation | 0.0004 | 0.0080 | 49.5829 | 0.0130 | 9.3353 |
| Hits | 50 | 34 | 0 | 0 | 42 |
| DLHS | | | | | |
| Best | 0.2281 | 0.0004 | 142.4515 | 1.3140 | 6.2444 |
| Standard deviation | 0.0130 | 0.0070 | 39.5126 | 3.1364 | 8.7724 |
| Hits | 50 | 50 | 0 | 32 | 44 |
| EHS | | | | | |
| Best | 0.0009 | 0.0001 | 97.2555 | 388.2894 | 4.0809 |
| Standard deviation | 0.0006 | 0.0007 | 7.4711 | 20.5949 | 5.2910 |
| Hits | 50 | 50 | 0 | 0 | 48 |
| AHS | | | | | |
| Best | **0.0003** | **0** | 96.4333 | 6.7471 | **3** |
| Standard deviation | **0.0006** | **0** | 0.2943 | 2.3279 | **0** |
| Hits | **50** | **50** | 0 | 0 | **50** |
| GAHS | | | | | |
| Best | 0.0263 | 0.0049 | **96.4034** | **0.0633** | **3** |
| Standard deviation | 0.0125 | 0.0021 | **0.7833** | **0.0337** | **0** |
| Hits | 50 | 49 | **0** | **44** | **50** |

[†]The accuracy tolerance for awarding a successful hit on the global minimum was decreased to 0.1 for 100-dimensional Rastrigin's function.
[%]Goldstein-Price was implemented in two dimensions as it is only defined in two.

one would expect that it would perform at least as well as AHS in all examples. However, as we see in Table 4, with Ackley's and Griewank's functions that this is not true. The reason for this is that AHS enters a slow convergence phase once the basin of attraction around local optima is found because of the drop in HM variance that causes the pitch adjustment operator to make smaller and smaller adjustments. This means that AHS requires several thousand iterations to find the exact optimum once it has found the basin of attraction. It therefore uses all the available iterations and keeps converging until the very end. If 30% of its available iterations is removed, as is done in GAHS, the final result will not be as accurate as it would have been otherwise.

Another weakness that affects the convergence speed is in the choice of the period length. The period length has to be long enough so that an accurate estimate can be made of the performance of the parameter set. This can only be an estimate since the true performance may only become apparent once an instance has fully converged, and we already know that AHS keeps converging even in the last few iterations. This means that our method of measuring the quality of a parameter set is inherently biased to those sets that perform well at the beginning of the optimisation process. This means that those sets with a small HMS and a large HMCR will frequently get chosen, causing valuable parameter sets that may later produce the best results to be discarded during the first round of eliminations.

An illustration of this effect is shown in Figure 7. The graph traces the convergence of GAHS over 500,000 iterations as it minimises Ackley's function. The fitness score of the best and the worst candidates that the optimiser currently has in the HM is recorded at each iteration and shown as two separate lines on the graph. We see a downward trend indicating convergence followed by discontinuous jumps that indicate the abandonment of one instance of AHS and the start of another. Note that there are three distinct convergence paths present in the first 150,000 iterations. These represent the three AHS instances that correspond to three parameter sets. The first one that clearly converges the quickest over the first 50,000 iterations corresponds to the parameter set where the HMS = 10. The two that follow are those that correspond to the HMS = 40 and HMS = 50 parameter sets. Since we know from the previous experiments that AHS performs better than GAHS and uses an HMS of 50, we know that GAHS would finally give the best results if the third instance was kept through both elimination rounds. However, due to its slow rate of convergence during the first 50,000 iterations, it is eliminated first, leaving a weaker parameter set to finally converge.

## 6. Conclusions

After investigating the results from comparing some of the best performing HS variants available today, we noticed that

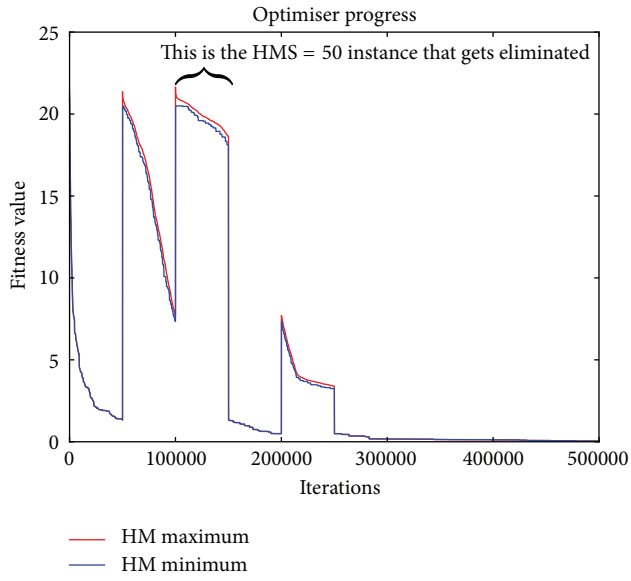


Figure 7: This graph traces the convergence of the GAHS algorithm over 500,000 iterations as it minimises Ackley's function. The fitness value, which is simply the candidate evaluated using Ackley's function, of the best and worst candidates in the HM is indidcated at the end of each iteration. The best candiate is the HM minimum indicated by the blue line; the worst candidate is the HM maximum indicated by the red line.

the overall quality of these algorithms was similar. We also noticed several weaknesses that prevent any one of them from being the best choice. Authors of these HS variants suggested parameter sets that are nearly optimal; however, some problems still required the parameters to be fine-tuned to a specific problem if good results were to be expected.

We suggested possible ways that some of these weaknesses can be addressed, and we proposed the generalised adaptive harmony search (GAHS) algorithm as a generalised version of AHS. GAHS performs well over a larger range of problems. By using 5 benchmark functions, we demonstrated that GAHS performs as well as AHS on most problems, and it can also produce good results on problems where AHS fails to find the global optimum.

However, some open questions still remain as a topic of future research. Our experiments clearly pointed out that there are categories of optimisation problems and that a parameter set that works well for one might fail completely for another one. For example, what makes Rastrigin's function so difficult to minimise using a parameter set that works well for all the other benchmark functions? What is special about Rastrigin's function that causes it to require a small HM to effectively optimise, and can this special attribute be detected before optimisation is attempted? A similar question can be asked about Rosenbrock's function. Why can none of the HM variants find the global optimum even when using a fine-tuned parameter set? Is there a weakness that is inherent in harmony search-based optimisers that make them poor optimisers for that category of problems to which Rosenbrock's function belongs, or can one augment harmony search in some way to address this weakness? Harmony search is still a relatively recent addition to the family of evolutionary algorithms, and finding answers to these type of questions is needed if harmony search is to become fully established as a leading heuristic optimiser.

## Acknowledgment

This work was supported by the Gachon University research fund of 2013 (GCU-2013-R084).

## References

[1] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: harmony search," *Simulation*, vol. 76, no. 2, pp. 60–68, 2001.

[2] O. M. Alia, R. Mandava, D. Ramachandram, and M. E. Aziz, "Dynamic fuzzy clustering using harmony search with application to image segmentation," in *Proceedings of the 9th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT '09)*, pp. 538–543, December 2009.

[3] J. Fourie, S. Mills, and R. Green, "Harmony filter: a robust visual tracking system using the improved harmony search algorithm," *Image and Vision Computing*, vol. 28, no. 12, pp. 1702–1716, 2010.

[4] J. Fourie, R. Green, and S. Mills, "Counterpoint harmony search: an accurate algorithm for the blind deconvolution of binary images," in *Proceedings of the International Conference on Audio, Language and Image Processing (ICALIP '10)*, pp. 1117–1122, Shanghai, China, November 2010.

[5] Z. W. Geem, *Recent Advances in Harmony Search Algorithm*, vol. 270 of *Studies in Computational Intelligence*, Springer, Berlin, Germany, 1st edition, 2010.

[6] Z. W. Geem, "Harmony search algorithm for solving Sudoku," in *Knowledge-Based Intelligent Information and Engineering Systems*, B. Apolloni, R. Howlett, and L. Jain, Eds., Lecture Notes in Computer Science, pp. 371–378, Springer, Berlin, Germany, 2010.

[7] Z. W. Geem, K. S. Lee, and Y. Park, "Application of harmony search to vehicle routing," *American Journal of Applied Sciences*, vol. 2, no. 12, pp. 1552–1557, 2005.

[8] Z. W. Geem and J.-Y. Choi, "Music composition using harmony search algorithm," in *Proceedings of the EvoWorkshops*, pp. 593–600, Springer, Berlin, Germany.

[9] M. Mahdavi, M. Fesanghary, and E. Damangir, "An improved harmony search algorithm for solving optimization problems," *Applied Mathematics and Computation*, vol. 188, no. 2, pp. 1567–1579, 2007.

[10] M. G. H. Omran and M. Mahdavi, "Global-best harmony search," *Applied Mathematics and Computation*, vol. 198, no. 2, pp. 643–656, 2008.

[11] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the International Conference on Neural Networks*, pp. 1942–1948, December 1995.

[12] M. Fesanghary, M. Mahdavi, M. Minary-Jolandan, and Y. Alizadeh, "Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 197, no. 33–40, pp. 3080–3091, 2008.

[13] Z. W. Geem, "Particle-swarm harmony search for water network design," *Engineering Optimization*, vol. 41, no. 4, pp. 297–311, 2009.

[14] H. Jiang, Y. Liu, and L. Zheng, "Design and simulation of simulated annealing algorithm with harmony search," in *Advances in Swarm Intelligence*, Y. Tan, Y. Shi, and K. Tan, Eds., Lecture Notes in Computer Science, pp. 454–460, Springer, Berlin, Germany, 2010.

[15] W. S. Jang, H. I. Kang, and B. H. Lee, "Hybrid simplex-harmony search method for optimization problems," in *Proceedings of IEEE Congress on Evolutionary Computation (CEC '08)*, pp. 4157–4164, June 2008.

[16] L. P. Li and L. Wang, "Hybrid algorithms based on harmony search and differential evolution for global optimization," in *Proceedings of the 1st ACM/SIGEVO Summit on Genetic and Evolutionary Computation (GEC '09)*, pp. 271–278, ACM, New York, NY, USA, June 2009.

[17] O. M. Alia and R. Mandava, "The variants of the harmony search algorithm: an overview," *Artificial Intelligence Review*, vol. 36, no. 1, pp. 49–68, 2011.

[18] S. Das, A. Mukhopadhyay, A. Roy, A. Abraham, and B. K. Panigrahi, "Exploratory power of the harmony search algorithm: analysis and improvements for global numerical optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 41, no. 1, pp. 89–106, 2011.

[19] C. M. Wang and Y. F. Huang, "Self-adaptive harmony search algorithm for optimization," *Expert Systems with Applications*, vol. 37, no. 4, pp. 2826–2837, 2010.

[20] H. Faure, "Good permutations for extreme discrepancy," *Journal of Number Theory*, vol. 42, no. 1, pp. 47–56, 1992.

[21] Q. K. Pan, P. N. Suganthan, J. J. Liang, and M. F. Tasgetiren, "A local-best harmony search algorithm with dynamic subpopulations," *Engineering Optimization*, vol. 42, no. 2, pp. 101–117, 2010.

[22] H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," *The Computer Journal*, vol. 3, no. 3, pp. 175–184, 1960.

[23] A. Törn and A. Pilinskas, *Global Optimization*, Lecture Notes in Computer Science 350, Springer, Berlin, Germany, 1989.

[24] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing*, vol. SECS28 of *The Kluwer International Series in Engineering and Computer Science*, Kluwer Academic Publishers, Boston, Mass, USA, 1987.

[25] J. Fourie, S. Mills, and R. Green, "Visual tracking using the harmony search algorithm," in *Proceedings of the 23rd International Conference Image and Vision Computing New Zealand (IVCNZ '08)*, Queenstown, New Zealand, November 2008.

[26] T. Niwa and M. Tanaka, "Analysis on the island model parallel genetic algorithms for the genetic drifts," in *Selected papers from the Second Asia-Pacific Conference on Simulated Evolution and Learning on Simulated Evolution and Learning (SEAL'98)*, vol. 98, pp. 349–356, Springer, London, UK, 1999.

[27] B. Artyushenko, "Analysis of global exploration of island model genetic algorithm," in *Proceedings of the 10th International Conference on Experience of Designing and Application of CAD Systems in Microelectronics (CADSM '09)*, pp. 280–281, February 2009.