Exploiting Gradient Information in Harmony Search

Bruno Figueira Lourenço
University of Brasília
Brasília, Distrito Federal - DF
brunofigueira@cic.unb.br

Marcelo Ladeira
University of Brasília
Brasília, Distrito Federal - DF
mladeira@unb.br

Abstract

Harmony Search (HS) is an optimization algorithm that mimicks the improvisation process of jazz musicians. It was originally conceived as an derivative-free method. But what if it is feasible to evaluate the gradient of the objective function? We believe that any optimization algorithm should use all information available. In this article, we describe an improvement of the HS algorithm when the gradient is indeed available.

Keywords: Harmony Search. Hybrid methods. Steepest Descent. Metaheuristics.

1 Introduction

Harmony Search (Geem et al., 2001; Lee and Geem, 2005; Geem, 2010) is a search technique for optimization problems that mimicks the creative process of musical improvisation. As such, it has found many applications in several engineering problems (Geem, 2010, 2006; Geem et al., 2008; Geem, 2009).

The original technique (Lee and Geem, 2005), as described by Geem, was advertised as a derivative-free search algorithm. But the question is: if the gradient of the objective function is indeed available, should not we try to exploit it?

The drawback of computing the gradient is an increase of the overall computational cost of the algorithm. There is also a certain risk of getting trapped in local minima. Fortunately, the random mechanisms of the Harmony Search algorithm should provide a protection against this trap.

So if the gradient is available and is feasible to be evaluated, we could use it to guide our search and to provide an useful hint that should improve the performance of the harmony search algorithm. In this article, we propose the use of the gradient information in the Harmony Search algorithm, present an analysis of the gains in performance and the associated overhead. We also use our new approach to find the minimum of well-known testbed functions

Section 2 presents an overview of the Harmony Search algorithm. Section 3 discusses the steepest descent method. Section 4 describes our approach to improve the Harmony Search algorithm by using the gradient of the objective function. Section 6 summarizes this work and describe future works.

2 Harmony Search

Phenomenon mimicking algorithms are common in the field of optimization. Genetic Algorithms (Goldberg, 1989), for instance, mimicks the phenomenon of natural selection and evolution. Particle Swarm Optimization (Kennedy and Eberhart, 1995) mimicks the swarm behaviour of certain animals and insects and tries to find a better "leader" for the swarm at each iteration.

In a similar fashion, Harmony Search (HS) mimicks the improvisation process of jazz musicians and tries to find the best harmony, i.e., the solution for a certain problem. Consider the problem of finding the minimum of a function $f(\vec{x})$ subject to $x_i \in X_i, i = 1,2,\ldots,n$, where X_i is the possible range for each variable with $x_i^L \leq x_i \leq x_i^U$, where x_i^L and x_i^U are the lower and upper bounds for each variable. HS work as follows:

1. Initialize a Harmony Memory (HM)
2. Improvise a new harmony vector from the HM
3. Update the HM with the new harmony vector.
4. Repeat steps 2 and 3 until a certain stopping criterion is satisfied.

The parameter Harmony Memory Size (HMS) represents the number of harmony vectors that the algorithm remembers simultaneously. Initially, the HM is filled with random harmonies (solutions) generated according to the bounds of the decision variables. At each iteration, we improvise a new harmony and if the new solutions is better than the worst in the HM, we add it to the HM and delete the worst harmony vector. Of course, the heart of the method lies in how to "improvise" a new harmony vector.

2.1 Improvisation

A new harmony vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ is generated from HM based on the following parameters: HMCR (Harmony Memory Considering Rate), PAR (Parameter Adjustment Rate) and BW (Bandwidth). So, for each x_i, we generate an uniform random variable u_1 between 0 and 1. If $u_1 \leq HMCR$, we randomly select an x_i from the HM. Otherwise, we choose an x_i from the entire feasible range, i.e. $[x_i^L, x_i^U]$

If we choose an x_i from the HM, we generate another uniform random variable u_2 between 0 and 1. If $u_2 \leq PAR$, we adjust the "pitch" of x_i by adding a small increment of α, where $\alpha = \text{BW} \times u_3$ and u_3 is an uniform random variable between -1 and 1.

Mahdavi et al. (Mahdavi et al., 2007) suggested that the PAR should increase linearly and BW should decrease exponentially after each iteration. Let MI be the maximum number of iterations, and k be the current iteration. At the k-th iteration we have:

$$\text{PAR}(k) = \text{PAR}_{\min} + (\text{PAR}_{\max} - \text{PAR}_{\min}) \times \frac{k}{\text{MI}} \tag{1}$$

$$\text{BW}(k) = \text{BW}_{\max} \exp\left[\ln\left(\frac{\text{BW}_{\min}}{\text{BW}_{\max}}\right)\frac{k}{\text{MI}}\right] \tag{2}$$

The entire algorithm is shown on Figure 1, where $u(a,b)$ denotes an uniform random variable between a and b. $\text{bounds}_{\min}(j)$ and $\text{bounds}_{\max}(j)$ denotes the lower and upper bounds for coordinate j.

3 Steepest Descent

Gradient based methods form the core of Nonlinear Programming and have been studied for decades by researchers in the field of optimization (Luenberger, 1984; Bertsekas, 1999; Nocedal and Wright, 2006). While it is true that sometimes it is not feasible to evaluate the gradient of a certain objective function, we should at least have ways of exploiting when it is indeed feasible.

In this work, we will only deal with the most basic form of gradient based methods, the Steepest Descent. Future work will focus on more sophisticated methods, such as BFGS (Bertsekas, 1999). First, we will present some basic notation. Let $f : \mathbb{R}^n \to \mathbb{R}$. The gradient of f at $\vec{x}$, denoted by $\nabla f(x)$ is the vector of first-order partial derivatives.

$$\nabla f(\vec{x})^T = \left(\frac{\partial f(\vec{x})}{x_1}, \frac{\partial f(\vec{x})}{x_2}, \ldots, \frac{\partial f(\vec{x})}{x_n}\right) \tag{3}$$

The gradient $\nabla f(\vec{x})$ of a function points in the direction of the greatest *increase*. So if we want to find a minimum of f, $-\nabla f(\vec{x})$ gives us a natural search direction, since the it points in the direction of the greatest *decrease*. Let $\vec{x}_0$ be a initial guess on the minimum of $f(\vec{x})$. The steepest descent update is given by:

$$\vec{x}^{k+1} = \vec{x}^k - \alpha^k \nabla f(\vec{x}^k)^T \tag{4}$$

Where α^k is chosen to minimize $\phi(\alpha^k) = f(\vec{x}^k - \alpha^k \nabla f(\vec{x}^k)^T)$. Therefore, to find a suitable α^k we must perform an unidimensional line search. For example, let $f(\vec{x}) = x_1^2 + x_2^2$. Of course, the minimum is 0 at $(0,0)$. But suppose that we would like to find the minimum with the steepest descent. Let $\vec{x}_0 = (4,2)$, so:

$$\phi(\alpha^0) = (4 - 8\alpha^0)^2 + (2 - 4\alpha^0)^2 \tag{5}$$

Input: $f(\vec{x})$, the bounds for each variable and the set of parameters.
1: **for** $i = 1$ to HMS **do**
2: Generate random solution and append to HM
3: **end for**
4: **for** $k = 1$ to MI **do**
5: Let $\vec{x} = (x_1, x_2, \ldots, x_n)$ be a new solution.
6: **for** $i = 1$ to n **do**
7: **if** $u(0,1) \leq$ HMCR **then**
8: Select a random solution y from HM
9: $x_i \leftarrow y_i$
10: PAR $\leftarrow$ PAR(k), as given by Equation 1
11: **if** $u(0,1) \leq$ PAR **then**
12: BW $\leftarrow$ BW(k), as given by Equation 2
13: $x_i \leftarrow x_i + u(-1,1) \times$ BW
14: **end if**
15: **else**
16: $x_i \leftarrow u(\text{bounds}_{\min}(i), \text{bounds}_{\max}(i))$,
17: **end if**
18: **end for**
19: Let $\vec{w}$ be the worst solution from HM
20: **if** $f(\vec{x}) < f(\vec{w})$ **then**
21: Delete $\vec{w}$ and append $\vec{x}$
22: **end if**
23: **end for**
24: **return** The best solution from HM

Figure 1: State-of-the-Art Harmony Search.

The minimum of $\phi(\alpha^0)$ is 0 at $\alpha^0 = 0.5$. So $\vec{x}^1 = (4,2) - 0.5(8,4) = (0,0)$. In this case, we have found the minimum in just one step, but this seldom happens. Also, we performed an exact line search and found the *exact* α^k that minimizes $\phi(\alpha^k)$. In real applications, it is usually too costly to perform the line search exactly at each iteration.

A more common approach is to find an α^k that provides a sufficient decrease of the objective function. However, it is interesting to note that just asking for an α^k such as $f(\vec{x}^k - \alpha^k \nabla f(\vec{x}^k)^T) < f(x^k)^T$ is not enough to ensure convergence to a minimum. The strong Wolfe's conditions (Nocedal and Wright, 2006), for example, ask for an α^k that satisfies:

$$\phi(\alpha^k) \leq \phi(0) + c_1 \alpha^k \nabla f(\vec{x}^k)^T p^k \tag{6}$$

$$|\nabla f(\vec{x}^k + \alpha^k p^k)^T p^k| \leq c_2 |\nabla f(\vec{x}^k)^T p^k| \tag{7}$$

Where p^k is a search direction and $0 < c_1 < c_2 < 1$. Since we are using the steepest descent, we have $p^k = -\nabla f(\vec{x}^k)^T$. In practice, c_1 is chosen to be small, for example, $c_1 = 10^{-4}$ and c_2 is usually chosen in $[0.1, 0.9]$ (Nocedal and Wright, 2006). Given an initial s and $0 < \beta < 1$, a simple but efficient way of performing the line search is to try the stepsizes $\{s, s\beta, s\beta^2, \ldots\}$ until Equations 6 and 7 are satisfied.

4 Harmony Search and the Steepest Descent

Building on the Harmony Search algorithm described in Figure 1, our idea is to give an useful tip to the algorithm by performing one step of the steepest descent after the end of the "improvisation".

But since it would be costly to perform this at each iteration, we introduce a parameter called "GU" (Gradient Usage) such as $0 < GU < 1$. So with probability GU we update the current solution with gradient information. In our test implementation, the line search is performed using the strong Wolfe's condition, but weaker conditions such as Armijo's rule (Nocedal and Wright, 2006) could be used if the evaluation of the gradient is feasible but expensive. Our improved algorithm is shown on Figure 2.

5 Computational Experiments

In this section we show a few tests and examples of our improved algorithm. Most of the examples were taken from (Mahdavi et al., 2007; Lee and Geem, 2005). We implemented our algorithm in the Python programming language. The line search method was provided by the optimization package of SciPy (Jones et al., 01) and it returns an step size that satisfies the strong Wolfe's conditions.

5.1 Rosenbrock Function

$$f(x,y) = 100(y-x^2)^2 + (1-x)^2 \tag{8}$$

The Rosenbrock Function, also known as the "banana function", is a well known test case for unconstrained optimization. It is very hard for gradient based methods to perform well because its global minimum is hidden inside a long narrow valley. Its global minimum is 0 at $(1,1)$. In our tests, we used bounds between -10 and 10 for both variables.

It is true that we are using the gradient in our improved algorithm, but due to the stochastic mechanisms we are able to explore several directions at the same time so we actually find the global minimum very fast. Reportedly, 50000 iterations were needed to find the soluction vector $x = (0.1000000000E+01, 0.1000002384E+01)$ with the original algorithm (Lee and Geem, 2005). Our own implementation, with dynamic PAR and BW, after 100 trials of 22000 iterations each, found an average mininum of $x = (1.00126167, 1.0027055)$.

We do not know for sure how the algorithm described in (Lee and Geem, 2005) was implemented, but it seems clear to us that at least 50000 evaluations of the objective function were needed, since at each step we have check if the new harmony is better than the worst harmony in the Harmony Memory. Our own implementation of Harmony Search algorithm always evaluates the objective function MI + HMS times, since we evaluate the function once at each iteration and once for each harmony in the HM . The number of function evaluations in this example was 22020 for all 100 trials.

In our improved algorithm, after 100 trials of 5000 iterations each, we found an average minimum of $x = (1.004740541.01163841)$, with $GU = 0.2$. The number of function evaluations in this case depends on the way the line search is performed. In average, using the strong Wolfe's conditions as line search method, the objective function was evaluated 12288 times and the gradient was evaluated 2197 times.

The parameters were the same for both the HS and the HS+Gradient Descent algorithms. HMCR = 0.90, HMS = 20, $\text{PAR}_{\text{min}} = 0.35$, $\text{PAR}_{\text{max}} = 0.99$, $\text{BW}_{\text{min}} = 0.01$ and $\text{BW}_{\text{max}} = 10$.

Input: $f(x)$, the bounds for each variable and the search parameters.

for $i = 1$ to HMS **do**
  Generate random solution and append to HM
end for
for $k = 1$ to MI **do**
  Let $x = (x_1, x_2, \ldots, x_n)$ be a new solution.
  for $i = 1$ to n **do**
    if $u(0,1) \leq$ HMCR **then**
      Select a random solution y from HM
      $x_i \leftarrow y_i$
      PAR $\leftarrow$ PAR(k), as given by Equation 1
      if $u(0,1) \leq PAR$ **then**
        BW $\leftarrow$ BW(k), as given by Equation 2
        $x_i \leftarrow x_i + u(-1,1) \times$ BW
      end if
    else
      $x_i \leftarrow u(\text{bounds}_{\min}(i), \text{bounds}_{\max}(i))$,
    end if
  end for
  if $(u(0,1) <$ GU **then**
    $\vec{p}_k \leftarrow -\nabla f(\vec{x})$
    Perform line search with $\vec{p}_k$ as search direction and find an α that satisfies the strong Wolfe's conditions.
    $\vec{x} \leftarrow \vec{x} + \alpha \vec{p}_k$
  end if
  Let $\vec{w}$ be the worst solution from HM
  if $f(\vec{x}) < f(\vec{w})$ **then**
    Delete $\vec{w}$ and append $\vec{x}$
  end if
end for
return The best solution from HM

Figure 2: The improved harmony search.

	F. Evaluations		G. Evaluations		Minimum		
	HS	HS+GD	HS	HS+GD	HS	HS+GD	Exact
Rosenbrock	22020	12288	0	2197	0.000162	0.002128	0
Powell Quartic	30007	15683	0	3045	1.34×10^{-7}	4.29×10^{-9}	0
Eason and Fenton	807	265	0	103	1.7441	1.7441	1.74

Table 1: Results for the test problems

5.2 Powell Quartic Function

$$f(x_1,x_2,x_3,x_4) = (x_1+10x_2)^2 + 5(x_3-x_4)^2 + (x_2-2x_3)^4 + 10(x_1-x_4)^4$$

This functions appears in (Powell, 1962; Lee and Geem, 2005) and is a standard benchmark of unconstrained optimization algorithms. Its global minimum is 0 at $(0,0,0,0)$. In our tests, we used bounds between -5 and 5 for all variables.

Using the original Harmony Search algorith, 100000 iterations were needed to find the minimum $x = (-0.0008828875, 0.0000882906, -0.0004372409, -0.0004372455)$.

With our own implementation, with dynamic BW and PAR, after 100 trials of 30000 iterations each, we found an average minimum of $x = (0.00511132, -0.00051472, 0.00327625, 0.00343767)$. The number of function evaluations after each trial was 30007.

After 100 trials of 5000 iterations each, the average minimum found with our improved method was $x = (-1.13181229e-03, 1.17699481e-04, -8.65751178e-05, -6.54175718e-05)$. The average number of function evaluations was 15683 and the number of gradient evaluations was 3045.

The parameters were the same for both the HS and the HS+Gradient Descent algorithms. HMCR = 0.90, HMS = 7, $\text{PAR}_{\text{min}} = 0.35$, $\text{PAR}_{\text{max}} = 0.99$, $\text{BW}_{\text{min}} = 0.01$ and $\text{BW}_{\text{max}} = 10$. It seems clear that in this example, our algorithm compares favorably against the canonical algorithm in all aspects.

5.3 Eason and Fenton's gear train inertia function

$$f(x_1,x_2) = \frac{1}{10}\left\{12 + x_1^2 + \frac{1+x_2^2}{x_1^2} + \frac{x_1^2x_2^2+100}{(x_1x_2)^4}\right\} \tag{9}$$

This function was also used in (Lee and Geem, 2005) to test the original Harmony Search algorithm. In our tests, we used bounds between 0 and 10 for both variables.

With our own implementation, with dynamic BW and PAR, after 100 trials of 800 iterations each, we found an average minimum of $x = (1.74354618, 2.02984523)$. 807 function evaluations were needed at each trial.

After 100 trials of 150 iterations each, the average minimum found with our improved method was $x = (1.74362659, 2.0307607)$. The average number of function evaluations was 265 and the number of gradient evaluations was 103.

The parameters were the same for both the HS and the HS+Gradient Descent algorithms. HMCR = 0.90, HMS = 7, $\text{PAR}_{\text{min}} = 0.35$, $\text{PAR}_{\text{max}} = 0.99$, $\text{BW}_{\text{min}} = 0.01$ and $\text{BW}_{\text{max}} = 10$. It seems clear that in this example, our algorithm compares favorably against the canonical algorithm in all aspects.

5.4 The GU parameter

The choice of the GU parameter is problem-specific, but there are a few guidelines. A high GU will almost certainly increase the overall running time of the algorithm due to an increase in the number of

GU	F. Evaluations	G. Evaluations	Minimum
0.1	8606	1090	1.162883×10^{-2}
0.2	12288	2197	2.128×10^{-3}
0.3	15870	3261	5.698240×10^{-4}
0.4	19580	4351	2.837852×10^{-8}
0.5	23252	5442	2.180408×10^{-8}
0.6	27048	6547	9.345794×10^{-9}
0.7	30731	7642	6.060825×10^{-9}
1.0	41900	10900	1.620176×10^{-9}

Table 2: Different choices of GU and the minimum of the Rosenbrock Function

function and gradient evaluations, but it will make the algorithm more sensitive when the harmonies are near to an optimum of the objective function thus converging quickier.

For the problems shown in this paper our choice of $GU = 0.2$ worked quite well. But what would happen if we use different values? Table 2 shows how our improved algorithm performs with other choices of GU for the Rosenbrock function. For each GU value, we performed 100 trials of 5000 iterations each and then we calculated the average numer of function and gradient evaluations. It is interesting to note that even when we use the gradient at each iteration ($GU = 1$), the algorithm's stochastic mechanisms prevent us from getting trapped in local minima.

6 Conclusion

This work proposes an improvement in the original Harmony Search algorithm (Geem, 2010). By exploiting the information provided by the gradient of the objective function, we can give an useful tip to the Harmony Search algorithm and, hopefully, achieve better results.

We introduced a new parameter called GU (*Gradient Usage*). At the end of each improvisation, with probability GU, we perform one step of the steepest descent. In our implementation, we used the strong Wolfe's conditions to perform the line search and to find a suitable α for the steepest descent.

Three benchmark functions were presented to demonstrate the effectiveness of our method and to show that our improved algorithm can achieve better results than the original one. It is also interesting to note that in all three functions, our algorithm needed far less iterations.

As a future work, we can explore other gradient based methods such as the Quasi-Newton methods (BFGS and DFP, for example). We can also explore and analyse with detail other line search methods that could be used when it is expensive to evaluate the gradient.

References

Bertsekas, D. P. (1999). *Nonlinear Programming*. Athena Scientific, 2 edition.

Geem, Z. (2010). State-of-the-Art in the structure of harmony search algorithm. In *Recent Advances In Harmony Search Algorithm*, pages 1–10.

Geem, Z., Kim, J., and Loganathan, G. (2001). A new heuristic optimization algorithm: Harmony search. *SIMULATION*, 76(2):60–68.

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

Geem, Z. W. (2009). Multiobjective optimization of Time-Cost trade-off using harmony search. *Journal of Construction Engineering and Management*, 1(1).

Geem, Z. W., Fesanghary, M., Choi, J., Saka, M. P., Williams, J. C., Ayvaz, M. T., Li, L., Ryu, S., and Vasebi, A. (2008). Recent advances in harmony search. In *Advances in Evolutionary Algorithms*, pages 127–142. Witold Kosiński.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1 edition.

Jones, E., Oliphant, T., Peterson, P., et al. (2001–). SciPy: Open source scientific tools for Python.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. volume 4, pages 1942–1948.

Lee, K. S. and Geem, Z. W. (2005). A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 194(36-38):3902–3933.

Luenberger, D. G. (1984). *Linear and Nonlinear Programming, Second Edition*. Springer, 2 edition.

Mahdavi, M., Fesanghary, M., and Damangir, E. (2007). An improved harmony search algorithm for solving optimization problems. *Applied Mathematics and Computation*, 188(2):1567–1579.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer, 2 edition.

Powell, M. J. D. (1962). An iterative method for finding stationary values of a function of several variables. *The Computer Journal*, 5(2):147–151.