Harmony Search*

Jason Brownlee
jasonb@CleverAlgorithms.com
The Clever Algorithms Project
http://www.CleverAlgorithms.com

November 22, 2010
Technical Report: CA-TR-20101122-1

Abstract

The Clever Algorithms project aims to describe a large number of Artificial Intelligence algorithms in a complete, consistent, and centralized manner, to improve their general accessibility. The project makes use of a standardized algorithm description template that uses well-defined topics that motivate the collection of specific and useful information about each algorithm described. This report describes the Harmony Search algorithm using the standardized algorithm description template.

Keywords: `Clever, Algorithms, Description, Optimization, Harmony, Search`

1 Introduction

The Clever Algorithms project aims to describe a large number of algorithms from the fields of Computational Intelligence, Biologically Inspired Computation, and Metaheuristics in a complete, consistent and centralized manner [1]. The project requires all algorithms to be described using a standardized template that includes a fixed number of sections, each of which is motivated by the presentation of specific information about the technique [2]. This report describes the Harmony Search algorithm using the standardized algorithm description template.

2 Name

Harmony Search, HS

3 Taxonomy

Harmony Search belongs to the fields of Computational Intelligence and Metaheuristics.

4 Inspiration

Harmony Search was inspired by the improvisation of Jazz musicians. Specifically, the process by which the musicians (who may have never played together before) rapidly refine their individual improvisation though variation resulting in an aesthetic harmony.

5 Metaphor

Each musician corresponds to an attribute in a candidate solution from a problem domain, and each instrument's pitch and range corresponds to the bounds and constraints on the decision variable. The harmony between the musicians is taken a complete candidate solution at a given time, and the audiences aesthetic appreciation of the harmony represent the problem specific cost function. The musicians seek harmony over time through small variations and improvisations, which results in an improvement against the cost function.

6 Strategy

The information processing objective of the technique is to use good candidate solutions already discovered to influence the creation of new candidate solutions toward locating the problems optima. This is achieved by stochastically creating candidate solutions in a step-wise manner, where each component is either drawn randomly from a memory of high-quality solutions, adjusted from the memory of high-quality solution, or assigned randomly within the bounds of the problem. The memory of candidate solutions is initially random, and a greedy acceptance criteria is used to admit new candidate solutions only if they have an improved objective value, replacing an existing member.

7 Procedure

Algorithm 1 provides a pseudo-code listing of the Harmony Search algorithm for minimizing a cost function. The adjustment of a pitch selected from the harmony memory is typically linear, for example for continuous function optimization:

$$x\prime \leftarrow x + range \times \epsilon \tag{1}$$

where $range$ is a the user parameter (pitch bandwidth) to control the size of the changes, and ϵ is a uniformly random number $\in [-1, 1]$.

8 Heuristics

- Harmony Search was designed as a generalized optimization method for continuous, discrete, and constrained optimization and has been applied to numerous types of optimization problems.
- The harmony memory considering rate (HMCR) $\in [0, 1]$ controls the use of information from the harmony memory or the generation of a random pitch. As such, it controls the rate of convergence of the algorithm and is typically configured $\in [0.7, 0.95]$.
- The pitch adjustment rate (PAR) $\in [0, 1]$ controls the frequency of adjustment of pitches selected from harmony memory, typically configured $\in [0.1, 0.5]$. High values can result in the premature convergence of the search.
- The pitch adjustment rate and the adjustment method (amount of adjustment or fret width) are typically fixed, having a linear effect through time. Non-linear methods have been considered, for example refer to Geem [8].
- When creating a new harmony, aggregations of pitches can be taken from across musicians in the harmony memory.
- The harmony memory update is typically a greedy process, although other considerations such as diversity may be used where the most similar harmony is replaced.

Algorithm 1: Pseudo Code for the Harmony Search algorithm.

Input: $Pitch_{num}$, $Pitch_{bounds}$, $Memory_{size}$, $Consolidation_{rate}$, $PitchAdjust_{rate}$, $Improvisation_{max}$
Output: $Harmony_{best}$

```
Harmonies ← InitializeHarmonyMemory(Pitch_num, Pitch_bounds, Memory_size);
EvaluateHarmonies(Harmonies);
for i to Improvisation_max do
    Harmony ← 0;
    foreach Pitch_i ∈ Pitch_num do
        if Rand() ≤ Consolidation_rate then
            RandomHarmony^i_pitch ← SelectRandomHarmonyPitch(Harmonies, Pitch_i);
            if Rand() ≤ PitchAdjust_rate then
                Harmony^i_pitch ← AdjustPitch(RandomHarmony^i_pitch);
            else
                Harmony^i_pitch ← RandomHarmony^i_pitch;
            end
        else
            Harmony^i_pitch ← RandomPitch(Pitch_bounds);
        end
    end
    EvaluateHarmonies(Harmony);
    if Cost(Harmony) ≤ Cost(Worst(Harmonies)) then
        Worst(Harmonies) ← Harmony;
    end
end
return Harmony_best;
```

9 Code Listing

Listing 1 provides an example of the Harmony Search algorithm implemented in the Ruby Programming Language. The demonstration problem is an instance of a continuous function optimization that seeks $minf(x)$ where $f = \sum_{i=1}^{n} x_i^2$, $-5.0 \leq x_i \leq 5.0$ and $n = 3$. The optimal solution for this basin function is $(v_0, \ldots, v_{n-1}) = 0.0$. The algorithm implementation and parameterization are based on the description by Yang [9], with refinement from Geem [8].

```ruby
def objective_function(vector)
  return vector.inject(0.0) {|sum, x| sum + (x ** 2.0)}
end

def random_variable(min, max)
  return min + ((max - min) * rand())
end

def random_vector(search_space)
  return Array.new(search_space.length) {|i| random_variable(search_space[i][0],
      search_space[i][1]) }
end

def create_random_harmony(search_space)
  harmony = {}
  harmony[:vector] = random_vector(search_space)
  harmony[:fitness] = objective_function(harmony[:vector])
  return harmony
end
```

```ruby

def initialize_harmony_memory(search_space, memory_size)
  memory = Array.new(memory_size * 3){ create_random_harmony(search_space) }
  memory.sort!{|x,y| x[:fitness]<=>y[:fitness]}
  memory = memory[0...memory_size]
  return memory
end

def create_harmony(search_space, memory, consideration_rate, adjust_rate, range)
  vector = Array.new(search_space.length)
  search_space.length.times do |i|
    if rand() < consideration_rate
      value = memory[rand(memory.size)][:vector][i]
      value = value + range * random_variable(-1.0, 1.0) if rand() < adjust_rate
      value = search_space[i][0] if value < search_space[i][0]
      value = search_space[i][1] if value > search_space[i][1]
      vector[i] = value
    else
      vector[i] = random_variable(search_space[i][0], search_space[i][1])
    end
  end
  return {:vector=>vector}
end

def search(search_space, max_iter, memory_size, consideration_rate, adjust_rate, range)
  memory = initialize_harmony_memory(search_space, memory_size)
  best = memory.first
  max_iter.times do |iter|
    harmony = create_harmony(search_space, memory, consideration_rate, adjust_rate, range)
    harmony[:fitness] = objective_function(harmony[:vector])
    best = harmony if harmony[:fitness] < best[:fitness]
    memory << harmony
    memory.sort!{|x,y| x[:fitness]<=>y[:fitness]}
    memory.delete_at(memory.length-1)
    puts " > iteration=#{iter}, fitness=#{best[:fitness]}"
  end
  return best
end

if __FILE__ == $0
  problem_size = 3
  search_space = Array.new(problem_size) {|i| [-5, 5]}
  memory_size = 20
  consideration_rate = 0.95
  adjust_rate = 0.7
  range = 0.05
  max_iter = 500

  best = search(search_space, max_iter, memory_size, consideration_rate, adjust_rate, range)
  puts "done! Solution: f=#{best[:fitness]}, s=#{best[:vector].inspect}"
end
```

Listing 1: Harmony Search algorithm in the Ruby Programming Language

10 References

10.1 Primary Sources

Geem, et al. proposed the Harmony Search algorithm in 2001 [3].

10.2 Learn More

A book on Harmony Search, edited by Geem provides a collection of papers on the technique and its applications [5], chapter 1 provides a useful summary of the method heuristics for its configuration [9]. Similarly a second edited volume by Geem focuses on studies that provide more advanced applications of the approach [7], and chapter 1 provides a detailed walkthrough of the technique itself [8]. Geem also provides a treatment of Harmony Search applied to the optimal design of water distribution networks [6] and edits yet a third volume on papers related to the application of the technique to structural design optimization problems [4].

11 Conclusions

This report described the Harmony Search algorithm using the standardized algorithm description template.

12 Contribute

Found a typo in the content or a bug in the source code? Are you an expert in this technique and know some facts that could improve the algorithm description for all? Do you want to get that warm feeling from contributing to an open source project? Do you want to see your name as an acknowledgment in print?

Two pillars of this effort are i) that the best domain experts are people outside of the project, and ii) that this work is subjected to continuous improvement. Please help to make this work less wrong by emailing the author 'Jason Brownlee' at `jasonb@CleverAlgorithms.com` or visit the project website at `http://www.CleverAlgorithms.com`.

References

[1] Jason Brownlee. The clever algorithms project: Overview. Technical Report CA-TR-20100105-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

[2] Jason Brownlee. A template for standardized algorithm descriptions. Technical Report CA-TR-20100107-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

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

[4] Zong Woo Geem, editor. *Harmony Search Algorithms for Structural Design Optimization*. Springer, 2009.

[5] Zong Woo Geem, editor. *Music-Inspired Harmony Search Algorithm: Theory and Applications*. Springer, 2009.

[6] Zong Woo Geem. *Optimal Design of Water Distribution Networks Using Harmony Search*. Lap Lambert Academic Publishing, 2009.

[7] Zong Woo Geem, editor. *Recent Advances in Harmony Search Algorithms*. Springer, 2010.

[8] Zong Woo Geem. *Recent Advances In Harmony Search Algorithms*, chapter State-of-the-Art in the Structure of Harmony Search Algorithm, pages 1–10. Springer, 2010.

[9] Xin-She Yang. *Music-Inspired Harmony Search Algorithm: Theory and Applications*, chapter Harmony Search as a Metaheuristic, pages 1–14. Springer, 2009.