Adaptive Random Search[*]

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

January 31, 2010
Technical Report: CA-TR-20100131-1

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 Adaptive Random Search algorithm using the standardized template.

Keywords: `Clever, Algorithms, Description, Adaptive, Random, Search, Optimization`

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 'Adaptive Random Search' algorithm using the standardized algorithm description template. Section 9 comments on the due diligence required for minimally investigate an algorithm for the Clever Algorithms project, and highlights some related algorithms discovered while preparing the description in this report.

2 Name

Adaptive Random Search, ARS, Adaptive Step Size Random Search, ASSRS, Variable Step-Size Random Search.

3 Taxonomy

The Adaptive Random Search algorithm belongs to the general set of approaches known as Stochastic Optimization and Global Optimization. It is a direct search method in that it does not require derivatives to navigate the search space. Adaptive Random Search is an extension of the Random Search and Localized Random Search algorithms.

4 Strategy

The Adaptive Random Search algorithm was designed to address the limitations of the fixed step size in the Localized Random Search algorithm. The strategy for Adaptive Random Search is to continually approximate the optimal step size required to reach the global optimum in the search space. This is achieved by trialling and adopting smaller or larger step sizes only if they result in an improvement in the search performance.

The Strategy of the Adaptive Step Size Random Search algorithm (the specific technique reviewed) is to trial a larger step in each iteration and adopt the larger step if it results in an improved result. Very large step sizes are trialled in the same manner although with a much lower frequency. This strategy of preferring large moves is intended to allow the technique to escape local optimal. Smaller step sizes are adopted if no improvement is made for an extended period.

5 Procedure

Algorithm 1 provides a pseudo-code listing of the Adaptive Random Search Algorithm for minimizing a cost function based on the specification for 'Adaptive Step-Size Random Search' by Schummer and Steiglitz [9].

6 Heuristics

- Adaptive Random Search was designed for continuous function optimization problem domains.
- Candidates with equal cost should be considered improvements to allow the algorithm to make progress across plateaus in the response surface.
- Adaptive Random Search may adapt the search direction in addition to the step size.
- The step size may be adapted for all parameters, or for each parameter individually.

7 Code Listing

Listing 1 provides an example of the Adaptive Random Search Algorithm implemented in the Ruby Programming Language, based on the specification for 'Adaptive Step-Size Random Search' by Schummer and Steiglitz [9]. In the example, the algorithm runs for a fixed number of iterations and returns the best candidate solution discovered. The example problem is an instance of a continuous function optimization that seeks $minf(x)$ where $f = \sum_{i=1}^{n} x_i^2$, $-5.0 < x_i < 5.0$ and $n = 2$. The optimal solution for this basin function is $(v_0, \ldots, v_{n-1}) = 0.0$.

```ruby
NUM_ITERATIONS = 1000
PROBLEM_SIZE = 2
SEARCH_SPACE = Array.new(PROBLEM_SIZE) {|i| [-5, +5]}
INITIAL_STEP_SIZE_FACTOR = 0.05
STEP_FACTOR_SMALL = 1.3
STEP_FACTOR_LARGE = 3
STEP_FACTOR_LARGE_MULTIPLE = 10
MAX_NO_IMPROVEMENTS = 30

def cost(candidate_vector)
  return candidate_vector.inject(0) {|sum, x| sum + (x ** 2.0)}
end

def random_solution(problemSize, searchSpace)
```

Algorithm 1: Pseudo Code Listing for the Adaptive Random Search Algorithm.

```
Input: $Iter_{max}$, ProblemSize, SearchSpace, $StepSize_{init}$, $StepSizeF_{small}$, $StepSizeF_{large}$,
       $StepSizeIter_{large}$, $NoChng_{max}$
Output: Current
$nochng_{count} \leftarrow 0$;
$step_{size} \leftarrow$ InitializeStepSize(SearchSpace, $StepSize_{init}$);
Current $\leftarrow$ RandomSolution(ProblemSize, SearchSpace);
foreach $iter_i \in Iter_{max}$ do
    $candidate_1 \leftarrow$ TakeStep(SearchSpace, Current, $step_{size}$);
    $largestep_{size} \leftarrow 0$;
    if $iter_i$ mod $StepSizeIter_{large}$ then
        $largestep_{size} \leftarrow step_{size} \times StepSizeF_{large}$;
    end
    else
        $largestep_{size} \leftarrow step_{size} \times StepSizeF_{small}$;
    end
    $candidate_2 \leftarrow$ TakeStep(SearchSpace, Current, $largestep_{size}$);
    if Cost($candidate_1$)$\leq$Cost(Current) or Cost($candidate_2$)$\leq$Cost(Current) then
        if Cost($candidate_2$)<Cost($candidate_1$) then
            Current $\leftarrow candidate_2$;
            $step_{size} \leftarrow largestep_{size}$;
        end
        else
            Current $\leftarrow candidate_1$;
        end
        $nochng_{count} \leftarrow 0$;
    end
    else
        $nochng_{count} \leftarrow nochng_{count} + 1$;
        if $nochng_{count} > NoChng_{max}$ then
            $nochng_{count} \leftarrow 0$;
            $step_{size} \leftarrow \frac{step_{size}}{StepSizeF_{small}}$
        end
    end
end
return Current;
```

```ruby
  return Array.new(problemSize) do |i|
    searchSpace[i][0] + ((searchSpace[i][1] - searchSpace[i][0]) * rand)
  end
end

def take_step(problemSize, searchSpace, currentPosition, stepSize)
  step = []
  problemSize.times do |i|
    max, min = currentPosition[i]+stepSize, currentPosition[i]-stepSize
    max = searchSpace[i][1] if max > searchSpace[i][1]
    min = searchSpace[i][0] if min < searchSpace[i][0]
    step << min + ((max - min) * rand)
  end
  return step
end

def search(numIterations, problemSize, searchSpace)
  stepSize = (searchSpace[0][1]-searchSpace[0][0]) * INITIAL_STEP_SIZE_FACTOR
  current, count = {}, 0
  current[:vector] = random_solution(problemSize, searchSpace)
  current[:cost] = cost(current[:vector])
  numIterations.times do |iter|
    step, biggerStep = {}, {}
    step[:vector] = take_step(problemSize, searchSpace, current[:vector], stepSize)
    step[:cost] = cost(step[:vector])
    biggerStepSize = stepSize * STEP_FACTOR_SMALL
    biggerStepSize = stepSize * STEP_FACTOR_LARGE if iter.modulo(STEP_FACTOR_LARGE_MULTIPLE)
    biggerStep[:vector] = take_step(problemSize, searchSpace, current[:vector], biggerStepSize)
    biggerStep[:cost] = cost(biggerStep[:vector])
    if step[:cost] <= current[:cost] or biggerStep[:cost] <= current[:cost]
      if biggerStep[:cost] < step[:cost]
        stepSize, current = biggerStepSize, biggerStep
      else
        current = step
      end
      count = 0
    else
      count += 1
      count, stepSize = 0, (stepSize/STEP_FACTOR_SMALL) if count >= MAX_NO_IMPROVEMENTS
    end
    puts " > iter #{(iter+1)}, cost=#{current[:cost]}, v=#{current[:vector].inspect}"
  end
  return current
end

best = search(NUM_ITERATIONS, PROBLEM_SIZE, SEARCH_SPACE)
puts "Done. Best Solution: cost=#{best[:cost]}, v=#{best[:vector].inspect}"
```

Listing 1: Adaptive Random Search Algorithm in the Ruby Programming Language

8 References

8.1 Primary Sources

Many works in the 1960s and 1970s experimented with variable step sizes for Random Search methods. Schummer and Steiglitz are commonly credited the adaptive step size procedure, which they called 'Adaptive Step-Size Random Search' [9]. Their approach only modifies the step size based on an approximation of the optimal step size required to reach the global optima. Kregting and White review adaptive random search methods and propose an approach called 'Adaptive Directional Random Search' that modifies both the algorithms step size and direction in response to the cost function [4].

8.2 Learn More

White reviews extensions to Rastrigin's 'Creeping Random Search' [7] (fixed step size) that use probabilistic step sizes drawn stochastically from uniform and probabilistic distributions [10]. White also reviews works that propose dynamic control strategies for the step size, such as Karnopp [3] who proposes increases and decreases to the step size based on performance over very small numbers of trials. Schrack and Choit review random search methods that modify their step size in order to approximate optimal moves while searching, including the property of reversal [8]. Masri, et al. describe an adaptive random search strategy that alternates between periods of fixed and variable step sizes [5].

9 Conclusions

This work described the Adaptive Random Search algorithm, specifically the seminal Adaptive Step Size Random Search algorithm that seeks an approximation of the optimal step size required to reach the global optimum. A feature of function optimization highlighted in this report was the need to differentiate between local and global optima, and optimization strategies that are aware of this differentiation. It is suggested that response surface modality and local and global optima and the difference between strategies that seek them be discussed in documentation that accompanies the algorithm descriptions, such as the introductory chapter of the proposed book.

9.1 Due Diligence

An aspect that was highlighted while preparing this report was the need for a clear definition of what constitutes 'due diligence' for investigating a an algorithm. Minimum requirements are needed to provide some assurance that both seminal and useful references are discovered and communicated in an algorithm description. Table 1 provides a listing of online sources that (as a minimum) must be consulted for a given algorithm to locate first-pass[1] references. Specifically the technique must be searched for by name and aliases against each of the listed search domains. Effort may be eased through use of tools such as *Google Custom Search*[2] that allow a number of sites to be searched in parallel for a given query.

Domain	**URL**
Google Web	`http://www.google.com`
Google Scholar	`http://scholar.google.com`
Google Books	`http://books.google.com`
IEEE Explore	`http://ieeexplore.ieee.org`
Springer Link	`http://www.springerlink.com`
ACM Digital Library	`http://portal.acm.org`
Scirus	`http://www.scirus.com`

Table 1: A listing of search domains to be consulted while researching for an algorithm description.

9.2 Related Approaches

Some related approaches discovered during the research for Adaptive Random Search include: *Pure Random Search* [6] which is a theoretical random search method, and *Sequential Random Search* that is a synonym for Random Search.

[1]Subsequent passes are generated by references collected from sources discovered in the first pass.
[2]Google Custom Search: `http://www.google.com/cse`

10 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 wrong by default. 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] Dean C. Karnopp. Random search techniques for optimization problems. *Automatica*, 1(2–3):111–121, 1963.

[4] J. Kregting and R. C. White. Adaptive random search. Technical Report TH-Report 71-E-24, Eindhoven University of Technology, Eindhoven, Netherlands, 1971.

[5] S. F. Masri, G. A. Bekey, and F. B. Safford. Global optimization algorithm using adaptive random search. *Applied Mathematics and Computation*, 7(4):353–376, 1980.

[6] Nitin R. Patell, Robert L. Smith, and Zelda B. Zabinsky. Pure adaptive search in monte carlo optimization. *Mathematical Programming*, 43(1-3):317–328, 1989.

[7] L. A. Rastrigin. The convergence of the random search method in the extremal control of a many parameter system. *Automation and Remote Control*, 24:1337–1342, 1963.

[8] Gnther Schrack and Mark Choit. Optimized relative step size random searches. *Mathematical Programming*, 10(1):230–244, 1976.

[9] M. Schumer and K. Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13(3):270–276, 1968.

[10] R. C. White. A survey of random methods for parameter optimization. *Simulation*, 17(1):197–205, 1971.