# Extremal Optimization*

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

November 16, 2010
Technical Report: CA-TR-20101116a-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 Extremal Optimization algorithm using the standardized algorithm template.

**Keywords:** `Clever, Algorithms, Description, Optimization, Extremal`

## 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 [7]. 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 [8]. This report describes the Extremal Optimization algorithm using the standardized algorithm template.

## 2 Name

Extremal Optimization, EO

## 3 Taxonomy

Extremal Optimization is a stochastic search technique that has properties of being a local and global search method. It is generally related to hill climbing algorithms and provides the basis for extensions such as Generalized Extremal Optimization.

## 4 Inspiration

Extremal Optimization is inspired by the Bak-Sneppen self-organized criticality model of co-evolution from the field of statistical physics. The self-organized criticality model suggests that some dynamical systems have a critical point as an attractor, whereby the systems exhibit periods of slow movement or accumulation followed by short periods of avalanche or instability. Examples of such systems include land formation, earthquakes, and the dynamics of sand piles. The Bak-Sneppen considers these dynamics in co-evolutionary systems and in the punctuated equilibrium model, which is described as long periods of status followed by short periods of extension and large evolutionary change.

## 5 Metaphor

The dynamics of the system result in the steady improvement of a candidate solution with sudden and large crashes in the quality of the candidate solution. These dynamics allow two main phases of activity in the system: 1) to exploit higher quality solutions in a local search like manner, and 2) escape possible local optima with a population crash and explore the search space for a new area of high quality solutions.

## 6 Strategy

The objective of the information processing strategy is to iteratively identify the worst performing components of a given solution and replace or swap them with other components. This is achieved through the allocation of cost to the components of the solution based on their contribution to the overall cost of the solution in the problem domain. Once components are assessed they can be ranked and the weaker components replaced or switched with a randomly selected component.

## 7 Procedure

Algorithm 1 provides a pseudo-code listing of the Extremal Optimization algorithm for minimizing a cost function. The deterministic selection of the worst component in the `SelectWeakComponent` function and replacement in the `SelectReplacementComponent` function is classical EO. If these decisions are probabilistic making use of $\tau$ parameter, this is referred to as $\tau$-Extremal Optimization.

## 8 Heuristics

- Extremal Optimization was designed for combinatorial optimization problems, although variations have been applied to continuous function optimization.
- The selection of the worst component and the replacement component each iteration can be deterministic or probabilistic, the latter of which is referred to as $\tau$-Extremal Optimization given the use of a $\tau$ parameter.
- The selection of an appropriate scoring function of the components of a solution is the most difficult part in the application of the technique.
- For $\tau$-Extremal Optimization, low $\tau$ values are used (such as $\tau \in [1, 2]$) have been found to be effective.

**Algorithm 1**: Pseudo Code for the Extremal Optimization algorithm.

**Input**: ProblemSize, $iterations_{max}$, $\tau$
**Output**: $S_{best}$

$S_{current} \leftarrow$ `CreateInitialSolution(ProblemSize)`;
$S_{best} \leftarrow S_{current}$;
**for** $i = 1$ **to** $iterations_{max}$ **do**
  **foreach** $Component_i \in S_{current}$ **do**
    $Component_i^{cost} \leftarrow$ `Cost(`$Component_i$, $S_{current}$`)`;
  **end**
  RankedComponents $\leftarrow$ `Rank(`$Si_{components}$`)`
  $Component_i \leftarrow$ `SelectWeakComponent(RankedComponents,` $Component_i$, $\tau$`)`;
  $Component_j \leftarrow$ `SelectReplacementComponent(RankedComponents,` $\tau$`)`;
  $S_{candidate} \leftarrow$ `Replace(`$S_{current}$, $Component_i$, $Component_j$`)`;
  **if** `Cost(`$S_{candidate}$`)` $\leq$ `Cost(`$S_{best}$`)` **then**
    $S_{best} \leftarrow S_{candidate}$;
  **end**
**end**
**return** $S_{best}$;

## 9 Code Listing

Listing 1 provides an example of the Extremal Optimization algorithm implemented in the Ruby Programming Language. The algorithm is applied to the Berlin52 instance of the Traveling Salesman Problem (TSP), taken from the TSPLIB. The problem seeks a permutation of the order to visit cities (called a tour) that minimized the total distance traveled. The optimal tour distance for Berlin52 instance is 7542 units.

The algorithm implementation is based on the seminal work by Boettcher and Percus [5]. A solution is comprised of a permutation of city components. Each city can potentially form a connection to any other city, and the connections to other cities ordered by distance may be considered its neighborhood. For a given candidate solution, the city components of a solution are scored based on the neighborhood rank of the cities to which they are connected: $fitness_k \leftarrow \frac{3}{r_i+r_j}$, where $r_i$ and $r_j$ are the neighborhood ranks of cities $i$ and $j$ against city $k$. A city is selected for modification probabilistically where the probability of selecting a given city is proportional to $n_i^{-\tau}$, where $n$ is the rank of city $i$.

An extension to the provided implementation would be to always replace the longest edge on the selected worst component city.

```ruby
def euc_2d(c1, c2)
  Math.sqrt((c1[0] - c2[0])**2.0 + (c1[1] - c2[1])**2.0).round
end

def cost(permutation, cities)
  distance =0
  permutation.each_with_index do |c1, i|
    c2 = (i==permutation.length-1) ? permutation[0] : permutation[i+1]
    distance += euc_2d(cities[c1], cities[c2])
  end
  return distance
end

def calculate_neighbour_rank(city_number, cities, ignore=[])
  neighbors = []
  cities.each_with_index do |city, i|
    next if i==city_number or ignore.include?(i)
```

```
    neighbor = {:number=>i}
    neighbor[:distance] = euc_2d(cities[city_number], city)
    neighbors << neighbor
  end
  neighbors.sort!{|x,y| x[:distance] <=> y[:distance]}
  return neighbors
end

def nearest_neighbor_solution(cities)
  perm = [rand(cities.length)]
  while perm.length < cities.length
    neighbors = calculate_neighbour_rank(perm.last, cities, perm)
    perm << neighbors.first[:number]
  end
  return perm
end

def get_edges_for_city(city_number, permutation)
  c1, c2 = -1, -1
  permutation.each_with_index do |c, i|
    if c == city_number
      c1 = (i==0) ? permutation.last : permutation[i-1]
      c2 = (i==permutation.length-1) ? permutation.first : permutation[i+1]
      break
    end
  end
  raise "error" if c1==-1 or c2==-1
  return [c1, c2]
end

def calculate_city_fitness(permutation, city_number, cities)
  c1, c2 = get_edges_for_city(city_number, permutation)
  neighbors = calculate_neighbour_rank(city_number, cities)
  n1, n2 = -1, -1
  neighbors.each_with_index do |neighbor,i|
    n1 = i+1 if neighbor[:number] == c1
    n2 = i+1 if neighbor[:number] == c2
    break if n1!=-1 and n2!=-1
  end
  return 3.0 / (n1.to_f + n2.to_f)
end

def calculate_city_fitnesses(cities, permutation)
  city_fitnesses = []
  cities.each_with_index do |city, i|
    city_fitness = {:number=>i}
    city_fitness[:fitness] = calculate_city_fitness(permutation, i, cities)
    city_fitnesses << city_fitness
  end
  city_fitnesses.sort!{|x,y| x[:fitness] <=> y[:fitness]}
  return city_fitnesses
end

def probabilistic_selection(ordered_components, tau)
  sum = 0.0
  ordered_components.each_with_index do |component, i|
    component[:prob] = (i.to_f+1.0)**(-tau)
    sum += component[:prob]
  end
  selected_city = -1
  selection = rand()
  ordered_components.each_with_index do |component, i|
    selection -= (component[:prob]/sum)
```

```ruby
    if selection<=0.0 or i==ordered_components.length-1
      selected_city = component[:number]
      break
    end
  end
  return selected_city
end

def select_replacement_city(cities, weak_component, tau)
  neighbors = calculate_neighbour_rank(weak_component, cities)
  return probabilistic_selection(neighbors, tau)
end

def select_weak_city(city_fitnesses, tau)
  return probabilistic_selection(city_fitnesses, tau)
end

def update_permutation(permutation, p1, p2)
  perm = Array.new(permutation)
  perm[p1...p2] = perm[p1...p2].reverse
  return perm
end

def search(cities, max_iter, tau)
  current = {:vector=>nearest_neighbor_solution(cities)}
  current[:cost] = cost(current[:vector], cities)
  best = current
  max_iter.times do |iter|
    city_fitnesses = calculate_city_fitnesses(cities, current[:vector])
    weak_c = select_weak_city(city_fitnesses, tau)
    replacement_c = select_replacement_city(cities, weak_c, tau)
    candidate = {}
    candidate[:vector] = update_permutation(current[:vector], weak_c, replacement_c)
    candidate[:cost] = cost(candidate[:vector], cities)
    current = candidate
    best = candidate if candidate[:cost] < best[:cost]
    puts " > iteration #{(iter+1)}, current=#{current[:cost]}, best=#{best[:cost]}"
  end
  return best
end

if __FILE__ == $0
  berlin52 = [[565,575],[25,185],[345,750],[945,685],[845,655],[880,660],[25,230],
   [525,1000],[580,1175],[650,1130],[1605,620],[1220,580],[1465,200],[1530,5],
   [845,680],[725,370],[145,665],[415,635],[510,875],[560,365],[300,465],
   [520,585],[480,415],[835,625],[975,580],[1215,245],[1320,315],[1250,400],
   [660,180],[410,250],[420,555],[575,665],[1150,1160],[700,580],[685,595],
   [685,610],[770,610],[795,645],[720,635],[760,650],[475,960],[95,260],
   [875,920],[700,500],[555,815],[830,485],[1170,65],[830,610],[605,625],
   [595,360],[1340,725],[1740,245]]
  max_iterations = 500
  tau = 1.2

  best = search(berlin52, max_iterations, tau)
  puts "Done. Best Solution: c=#{best[:cost]}, v=#{best[:vector].inspect}"
end
```

Listing 1: Extremal Optimization algorithm in the Ruby Programming Language

# 10 References

## 10.1 Primary Sources

Extremal Optimization was proposed as an optimization heuristic by Boettcher and Percus applied to graph partitioning and the Traveling Salesman Problem [5]. The approach was inspired by the Bak-Sneppen self-organized criticality model of co-evolution [2, 1].

## 10.2 Learn More

A number of detailed reviews of Extremal Optimization have been presented, including a review and studies by Boettcher and Percus [4], an accessible review by Boettcher [3], and a focused study on the Spin Glass problem by Boettcher and Percus [6].

# 11 Conclusions

This report described the Extremal Optimization algorithm using the standardized algorithm template.

The implementation of the $\tau$-Extremal Optimization algorithm in the Ruby programming language require some additional work to provide an effective demonstration.

# 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] Per Bak and Kim Sneppen. Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters*, 71:4083–4086, 1993.

[2] Per Bak, Chao Tang, and Kurt Wiesenfeld. Self-organized criticality: An explanation of the 1/f noise. *Physical Review Letters*, 59:381–384, 1987.

[3] S. Boettcher. Extremal optimization: heuristics via coevolutionary avalanches. *Computing in Science & Engineering*, 2(6):75–82, 2000.

[4] S. Boettcher and A. Percus. Natures way of optimizing. *Artificial Intelligence*, 119(1-2):275–286, 2000.

[5] Stefan Boettcher and Allon G. Percus. Extremal optimization: Methods derived from co-evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999.

[6] Stefan Boettcher and Allon G. Percus. Optimization with extremal dynamics. *Phys. Rev. Lett.*, 86:5211–5214, 2001.

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

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