# Iterated Local Search*

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

February 10, 2010
Technical Report: CA-TR-20100210-1

## 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 Iterated Local Search algorithm using the standardized template.

**Keywords:** `Clever, Algorithms, Description, Optimization, Iterated, Local, 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 [3]. 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 [5]. This report describes the Iterated Local Search algorithm using the standardized template. Section 9 suggests some areas for future consideration for the Clever Algorithms project that arose during the preparation of this report.

## 2 Name

Iterated Local Search, ILS

## 3 Taxonomy

Iterated Local Search is a Metaheuristic and a Global Optimization technique. It is an extension of Mutli Start Search and may be considered a parent of many two-phase search approaches such as Greedy Randomized Adaptive Search Procedure [4] and Variable Neighborhood Search [6].

## 4 Strategy

The objective of Iterated Local Search is to improve upon stochastic Mutli-Restart Search by sampling in the broader neighborhood of candidate solutions and using a Local Search technique to refine solutions to their local optima. Iterated Local Search explores a sequence of solutions created as perturbations of the current best solution, the result of which is refined using an embedded heuristic.

## 5 Procedure

Algorithm 1 provides a pseudo-code listing of the Iterated Local Search algorithm for minimizing a cost function.

**Algorithm 1**: Pseudo Code for the Iterated Local Search algorithm.

```
Input:
Output: $S_{best}$
$S_{best}$ ← ConstructInitialSolution();
$S_{best}$ ← LocalSearch();
SearchHistory ← $S_{best}$;
while ¬ StopCondition() do
    $S_{candidate}$ ← Perturbation($S_{best}$, SearchHistory);
    $S_{candidate}$ ← LocalSearch($S_{candidate}$);
    SearchHistory ← $S_{candidate}$;
    if AcceptanceCriterion($S_{best}$, $S_{candidate}$, SearchHistory) then
        $S_{best}$ ← $S_{candidate}$;
    end
end
return $S_{best}$;
```

## 6 Heuristics

- Iterated Local Search was designed for and has been predominately applied to discrete domains, such as combinatorial optimization problems.
- The perturbation of the current best solution should be in a neighborhood beyond the reach of the embedded heuristic and should not be easily undone.
- Perturbations that are too small make the algorithm too greedy, perturbations that are too large make the algorithm too stochastic.
- The embedded heuristic is most commonly a problem-specific local search technique.
- The starting point for the search may be a randomly constructed candidate solution, or constructed using a problem-specific heuristic (such as nearest neighbor).
- Perturbations can be made deterministically, although stochastic and probabilistic (adaptive based on history) are the most common.
- The procedure may store as much or as little history as needed to be used during perturbation and acceptance criteria. No history represents a random walk in a larger neighborhood of the best solution and is the most common implementation of the approach.

- The simplest and most common acceptance criteria is an improvement in the cost of constructed candidate solutions.

## 7 Code Listing

Listing 1 provides an example of the Iterated Local Search 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 Iterated Local Search runs for a fixed number of iterations. The implementation is based on a common algorithm configuration for the TSP, where a 'double-bridge move' (4-opt) is used as the perturbation technique, and a stochastic 2-opt is used as the embedded Local Search heuristic. The double-bridge move involves partitioning a permutation into 4 pieces (a,b,c,d) and putting it back together in a specific and jumbled ordering (a,d,c,b).

```ruby
MAX_ITERATIONS = 100
LOCAL_SEARCH_NO_IMPROVEMENTS = 50
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]]

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 random_permutation(cities)
  all = Array.new(cities.length) {|i| i}
  return Array.new(all.length) {|i| all.delete_at(rand(all.length))}
end

def stochastic_two_opt(permutation)
  perm = Array.new(permutation)
  c1, c2 = rand(perm.length), rand(perm.length)
  c2 = rand(perm.length) while c1 == c2
  c1, c2 = c2, c1 if c2 < c1
  perm[c1...c2] = perm[c1...c2].reverse
  return perm
end

def local_search(best, cities, maxNoImprovements)
  noImprovements = 0
  begin
    candidate = {}
    candidate[:vector] = stochastic_two_opt(best[:vector])
    candidate[:cost] = cost(candidate[:vector], cities)
    if candidate[:cost] < best[:cost]
      noImprovements, best = 0, candidate
    else
      noImprovements += 1
    end
  end until noImprovements >= maxNoImprovements
  return best
end

def double_bridge_move(perm)
  pos1 = 1 + rand(perm.length / 4)
  pos2 = pos1 + 1 + rand(perm.length / 4)
  pos3 = pos2 + 1 + rand(perm.length / 4)
  return perm[0...pos1] + perm[pos3..perm.length] + perm[pos2...pos3] + perm[pos1...pos2]
end

def perturbation(cities, best)
  candidate = {}
  candidate[:vector] = double_bridge_move(best[:vector])
  candidate[:cost] = cost(candidate[:vector], cities)
  return candidate
end

def search(cities, maxIterations, maxNoImprovementsLS)
  best = {}
  best[:vector] = random_permutation(cities)
  best[:cost] = cost(best[:vector], cities)
  best = local_search(best, cities, maxNoImprovementsLS)
  maxIterations.times do |iter|
    candidate = perturbation(cities, best)
    candidate = local_search(candidate, cities, maxNoImprovementsLS)
    best = candidate if best.nil? or candidate[:cost] < best[:cost]
    puts " > iteration #{(iter+1)}, best: c=#{best[:cost]}"
  end
  return best
end

best = search(BERLIN52, MAX_ITERATIONS, LOCAL_SEARCH_NO_IMPROVEMENTS)
puts "Done. Best Solution: c=#{best[:cost]}, v=#{best[:vector].inspect}"
```

Listing 1: Iterated Local Search algorithm in the Ruby Programming Language

## 8 References

### 8.1 Primary Sources

The definition and framework for Iterated Local Search was described by Stützle in his PhD dissertation [16]. Specifically he proposed constrains on what constitutes an Iterated Local Search algorithm as i) a single chain of candidate solutions, and ii) the method used to improve candidate solutions occurs within a reduced space by a black-box heuristic. Stützle does not take credit for the approach, instead highlighting specific instances of Iterated Local Search from the literature, such as 'iterated descent' [1], 'large-step Markov chains' [11], 'iterated Lin-Kernighan' [7], 'chained local optimization' [10], as well as [2] that introduces the principle, and [8] that summarized it (list taken from [12])

### 8.2 Learn More

Two early technical reports by Stützle that present applications of Iterated Local Search include a report on the Quadratic Assignment Problem [14], and another on the permutation flow shop problem [13]. Stützle and Hoos also published an early paper studying Iterated Local Search for to the TSP [15]. Lourenco, Martin, and Stützle provide a concise presentation of the technique, related techniques and the framework, much as it is presented in Stützle's dissertation [9]. The same author's also preset an authoritative summary of the approach and its applications as a book chapter [12].

## 9 Conclusions

This report described the Iterated Local Search technique as a proposed improvement on the multi start technique.

The investigation for this report highlighted the potential need for an explanation of exploration versus exploitation (also known as diversification versus intensification) in the introductory material provided with the algorithm descriptions, such as the introduction chapter of the book. The investigation also highlighted the 'Multi Start Strategy' (also known as Random Restart) that may be important to describe given the relationship of the strategy to Iterated Local Search and similar techniques.

A final point that was highlighted in the preparation of this report was the importance of being specific in the case studies for the intended audience [3]. For example, i) the 'primary sources' section of the description should contain those references that may be consulted and cited when discussing a given technique in a paper, and ii) the implementation could be taken and adapted for a specific problem when the technique is selected for an application or comparison. The descriptions are intended as jumpstart guides for specific techniques, designed to save time rather than be exhaustive.

## 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] E. B. Baum. Towards practical "neural" computation for combinatorial optimization problems. In *AIP conference proceedings: Neural Networks for Computing*, pages 53–64, 1986.

[2] J. Baxter. Local optima avoidance in depot location. *Journal of the Operational Research Society*, 32:815–819, 1981.

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

[4] Jason Brownlee. Greedy randomized adaptive search procedures. Technical Report CA-TR-20100209-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, February 2010.

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

[6] Jason Brownlee. Variable neighborhood search. Technical Report CA-TR-20100206-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, February 2010.

[7] D. S. Johnson. Local optimization and the travelling salesman problem. In *Proceedings of the 17th Colloquium on Automata, Languages, and Programming*, pages 446–461, 1990.

[8] D. S. Johnson and L. A. McGeoch. *Local Search in Combinatorial Optimization*, chapter The travelling salesman problem: A case study in local optimization, pages 215–310. John Wiley & Sons, 1997.

[9] H. R. Lourenco, O. Martin, and T. Stützle. A beginners introduction to iterated local search. In *Proceedings 4th Metaheuristics International Conference (MIC2001)*, 2001.

[10] O. Martin and S. W. Otto. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[11] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problems. *Complex Systems*, 5(3):299–326, 1991.

[12] Helena Ramalhinho-Loureno, Olivier C. Martin, and Thomas Stützle. *Handbook of Metaheuristics*, chapter Iterated Local Search, pages 320–353. Springer, 2003.

[13] T. Stützle. Applying iterated local search to the permutation flow shop problem. Technical Report AIDA9804, FG Intellektik, TU Darmstadt, 1998.

[14] T. Stützle. Iterated local search for the quadratic assignment problem. Technical Report AIDA-99-03, FG Intellektik, FB Informatik, TU Darmstadt, 1999.

[15] Thomas Stützle and Holger H. Hoos. Analyzing the run-time behaviour of iterated local search for the tsp. In *Proceedings III Metaheuristics International Conference*, 1999.

[16] Thomas G. Stützle. *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications.* PhD thesis, Darmstadt University of Technology, Department of Computer Science, 1998.