Guided Local Search*

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

February 4, 2010
Technical Report: CA-TR-20100204-1

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 described the Guided Local Search algorithm using the standardized template.

Keywords: `Clever, Algorithms, Description, Optimization, Guided, 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 [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 Guided Local Search algorithm in terms of the standardized template.

2 Name

Guided Local Search, GLS

3 Taxonomy

The Guided Local Search algorithm is a Metaheuristic and a Global Optimization algorithm that makes use of an embedded Local Search algorithm. It is an extension to Local Search algorithms such as Hill Climbing and is similar in strategy to the Tabu Search algorithm and the Iterated Local Search algorithm.

4 Strategy

The strategy for the Guided Local Search algorithm is to use penalties to encourage a Local Search technique to escape local optima and discover the global optima. A Local Search algorithm is run until it gets stuck in a local optima. The features from the local optima are evaluated and penalized, the results of which are used in an augmented cost function employed by the Local Search procedure. The Local Search is repeated a number of times using the last local optima discovered and the augmented cost function that guides exploration away from solutions with features present in discovered local optima.

5 Procedure

Algorithm 1 provides a pseudo-code listing of the Guided Local Search algorithm for minimization. The Local Search algorithm used by the Guided Local Search algorithm uses an augmented cost function in the form $h(s) = g(s) + \lambda \cdot \sum_{i=1}^{M} f_i$, where $h(s)$ is the augmented cost function, $g(s)$ is the problem cost function,λ is the 'regularization parameter' (a coefficient for scaling the penalties), s is a locally optimal solution of M features, and f_i is the i'th feature in locally optimal solution. The augmented cost function is only used by the local search procedure, the Guided Local Search algorithm uses the problem specific cost function without augmentation.

Penalties are only updated for those features in a locally optimal solution that maximize utility, updated by adding 1 to the penalty for the future (a counter). The utility for a feature is calculated as $U_{feature} = \frac{C_{feature}}{1+P_{feature}}$, where $U_{feature}$ is the utility for penalizing a feature (maximizing), $C_{feature}$ is the cost of the feature, and $P_{feature}$ is the current penalty for the feature.

Algorithm 1: Pseudo Code Listing for the Guided Local Search algorithm.

Input: $Iter_{max}$, λ
Output: S_{best}

$f_{penalties} \leftarrow 0$;
$S_{best} \leftarrow$ `RandomSolution()`;
foreach $Iter_i \in Iter_{max}$ **do**
  $S_{curr} \leftarrow$ `LocalSearch(`S_{best}, λ, $f_{penalties}$`)`;
  $f_{utilities} \leftarrow$ `CalculateFeatureUtilities(`S_{curr}, $f_{penalties}$`)`;
  $f_{penalties} \leftarrow$ `UpdateFeaturePenalties(`S_{curr}, $f_{penalties}$, $f_{utilities}$`)`;
  if `Cost(`S_{curr}`)` $\leq$ `Cost(`S_{best}`)` **then**
    $S_{best} \leftarrow S_{curr}$;
  end
end
return S_{best};

6 Heuristics

- The Guided Local Search procedure is independent of the Local Search procedure embedded within it. A suitable domain-specific search procedure should be identified and employed.

- The Guided Local Search procedure may need to be executed for thousands to hundreds-of-thousands of iterations, each iteration of which assumes a run of a Local Search algorithm to convergence.

- The algorithm was designed for discrete optimization problems where a solution is comprised of independently assessable 'features' such as Combinatorial Optimization, although it has been applied to continuous function optimization modeled as binary strings.
- The λ parameter is a scaling factor for feature penalization that must be in the same proportion to the candidate solution costs from the specific problem instance to which the algorithm is being applied. As such, the value for λ must be meaningful when used within the augmented cost function (such as when it is added to a candidate solution cost in minimization and subtracted from a cost in the case of a maximization problem).

7 Code Listing

Listing 1 provides an example of the Guided 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 implementation of the algorithm for the TSP was based on the configuration specified by Voudouris in [8]. A TSP-specific local search algorithm is used called 2-opt that selects two points in a permutation and reconnects the tour, potentially untwisting the tour at the selected points. The stopping condition for 2-opt was configured to be a fixed number of non-improving moves.

The equation for setting λ for TSP instances is $\lambda = \alpha \cdot \frac{cost(optima)}{N}$, where N is the number of cities, $cost(optima)$ is the cost of a local optimum found by a local search, and $\alpha \in (0, 1]$ (around 0.3 for TSP and 2-opt). The cost of a local optima was fixed to the approximated value of 15000 for the Berlin52 instance. The utility function for features (edges) in the TSP is $U_{edge} = \frac{D_{edge}}{1+P_{edge}}$, where U_{edge} is the utility for penalizing an edge (maximizing), D_{edge} is the cost of the edge (distance between cities) and P_{edge} is the current penalty for the edge.

```
NUM_ITERATIONS = 100
MAX_NO_MPROVEMENTS = 15
ALPHA = 0.3
LOCAL_SEARCH_OPTIMA = 15000.0
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]]

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

def random_permutation(cities)
  perm = Array.new(cities.length){|i|i}
  for i in 0...perm.length
    r = rand(perm.length-i) + i
    perm[r], perm[i] = perm[i], perm[r]
  end
  return perm
end

def 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 augmented_cost(permutation, penalties, cities, lambda)
  distance, augmented = 0, 0
  permutation.each_with_index do |c1, i|
    c2 = (i==permutation.length-1) ? permutation[0] : permutation[i+1]
    c1, c2 = c2, c1 if c2 < c1
    d = euc_2d(cities[c1], cities[c2])
    distance += d
    augmented += d + (lambda * (permutation[c1][c2]))
  end
  return distance, augmented
end

def local_search(current, cities, penalties, maxNoImprovements, lambda)
  current[:cost], current[:acost] = augmented_cost(current[:vector], penalties, cities, lambda)
  noImprovements = 0
  begin
    perm = {}
    perm[:vector] = two_opt(current[:vector])
    perm[:cost], perm[:acost] = augmented_cost(perm[:vector], penalties, cities, lambda)
    if perm[:acost] < current[:acost]
      noImprovements, current = 0, perm
    else
      noImprovements += 1
    end
  end until noImprovements >= maxNoImprovements
  return current
end

def calculate_feature_utilities(penalties, cities, permutation)
  utilities = Array.new(permutation.length,0)
  permutation.each_with_index do |c1, i|
    c2 = (i==permutation.length-1) ? permutation[0] : permutation[i+1]
    c1, c2 = c2, c1 if c2 < c1
    utilities[i] = euc_2d(cities[c1], cities[c2]) / (1.0 + penalties[c1][c2])
  end
  return utilities
end

def update_penalties!(penalties, cities, permutation, utilities)
  max = utilities.max()
  permutation.each_with_index do |c1, i|
    c2 = (i==permutation.length-1) ? permutation[0] : permutation[i+1]
    c1, c2 = c2, c1 if c2 < c1
    penalties[c1][c2] += 1 if utilities[i] == max
  end
  return penalties
end

def search(numIterations, cities, maxNoImprovements, lambda)
  best, current = nil, {}
  current[:vector] = random_permutation(cities)
  penalties = Array.new(cities.length){Array.new(cities.length,0)}
  numIterations.times do |iter|
    current = local_search(current, cities, penalties, maxNoImprovements, lambda)
    utilities = calculate_feature_utilities(penalties, cities, current[:vector])
    update_penalties!(penalties, cities, current[:vector], utilities)
    if(best.nil? or current[:cost] < best[:cost])
      best = current
```

```ruby
    end
    puts " > iteration #{(iter+1)}, best: c=#{best[:cost]}"
  end
  return best
end

lambda = ALPHA * (LOCAL_SEARCH_OPTIMA/BERLIN52.length)
best = search(NUM_ITERATIONS, BERLIN52, MAX_NO_MPROVEMENTS, lambda)
puts "Done. Best Solution: c=#{best[:cost]}, v=#{best[:vector].inspect}"
```

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

8 References

8.1 Primary Sources

Guided Local Search emerged from an approach called GENET, which is a connectionist approach to constraint satisfaction [15, 7]. Guided Local Search was presented by Voudouris and Tsang in a series of technical reports (that were later published) that described the technique and provided example applications of it to constraint satisfaction [11], combinatorial optimization [14, 13], and function optimization [12]. The seminal work on the technique was Voudouris' PhD dissertation [8].

8.2 Learn More

Voudouris and Tsang provide a high-level introduction to the technique [10], and a contemporary summary of the approach in Glover and Kochenberger's 'Handbook of metaheuristics' [9] that includes a review of the technique, application areas, and demonstration applications on a diverse set of problem instances. Mills, et al. elaborated on the approach, devising an 'Extended Guided Local Search' (EGLS) technique that added 'aspiration criteria' and random moves to the procedure [6], work which culminated in Mills' PhD dissertation [5]. Lau and Tsang further extended the approach by integrating it with a Genetic Algorithm, called the 'Guided Genetic Algorithm' (GGA) [4], that also culminated in a PhD dissertation by Lau [3].

9 Conclusions

This report described the Guided Local Search algorithm as a metaheuristic to manage local search procedures for combinatorial problem instances. A technique Local Search procedure developed used by Voudouris and Tsang for use with Guided Local Search was an approach called 'Fast Local Search' [8].

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] L. T. Lau. *Guided Genetic Algorithm.* PhD thesis, Department of Computer Science, University of Essex, 1999.

[4] T.L. Lau and E.P.K. Tsang. The guided genetic algorithm and its application to the general assignment problems. In *IEEE 10th International Conference on Tools with Artificial Intelligence (ICTAI'98)*, 1998.

[5] P. Mills. *Extensions to Guided Local Search.* PhD thesis, Department of Computer Science, University of Essex, 2002.

[6] Patrick Mills, Edward Tsang, and John Ford. Applying an extended guided local search on the quadratic assignment problem. *Annals of Operations Research*, 118:121–135, 2003.

[7] E. P. K Tsang and C. J. Wang. A generic neural network approach for constraint satisfaction problems. In Taylor G, editor, *Neural network applications*, pages 12–22, 1992.

[8] C Voudouris. *Guided local search for combinatorial optimisation problems.* PhD thesis, Department of Computer Science, University of Essex, Colchester, UK, July 1997.

[9] C. Voudouris and E. P. K. Tsang. *Handbook of metaheuristics*, chapter 7: Guided Local Search, pages 185–218. Springer, 2003.

[10] C. Voudouris and E.P.K. Tsang. Guided local search joins the elite in discrete optimisation. In *Proceedings, DIMACS Workshop on Constraint Programming and Large Scale Discrete Optimisation*, 1998.

[11] Chris Voudouris and Edward Tsang. The tunneling algorithm for partial csps and combinatorial optimization problems. Technical Report CSM-213, Department of Computer Science, University of Essex, Colchester, C04 3SQ, UK, 1994.

[12] Chris Voudouris and Edward Tsang. Function optimization using guided local search. Technical Report CSM-249, Department of Computer Science University of Essex Colchester, CO4 3SQ, UK, 1995.

[13] Chris Voudouris and Edward Tsang. Guided local search. Technical Report CSM-247, Department of Computer Science, University of Essex, Colchester, C04 3SQ, UK, 1995.

[14] Edward Tsang & Chris Voudouris. Fast local search and guided local search and their application to british telecoms workforce scheduling problem. Technical Report CSM-246, Department of Computer Science University of Essex Colchester CO4 3SQ, 1995.

[15] C. J. Wang and E. P. K. Tsang. Solving constraint satisfaction problems using neural networks. In *Proc. Second International Conference on Artificial Neural Networks*, pages 295–299, November 18–20, 1991.