Greedy Randomized Adaptive Search Procedure*

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

February 9, 2010
Technical Report: CA-TR-20100209-1

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 Greedy Randomized Adaptive Search Procedure using the standardized template.

Keywords: `Clever, Algorithms, Description, Optimization, Greedy, Randomized, Adaptive, Search, Procedure, GRASP`

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 [2]. 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 [3]. This report describes the Greedy Randomized Adaptive Search Procedure using the standardized template.

2 Name

Greedy Randomized Adaptive Search Procedure, GRASP

3 Taxonomy

The Greedy Randomized Adaptive Search Procedure is a Metaheuristic and Global Optimization algorithm, originally proposed for the Operations Research practitioners. The iterative application of an embedded Local Search technique relate the approach to Iterative Local Search and Multi-Start techniques.

4 Strategy

The objective of the Greedy Randomized Adaptive Search Procedure is to repeatedly sample stochastically greedy solutions, and then use a local search procedure to refine them to a local optima. The strategy of the procedure is centered on the stochastic and greedy step-wise construction mechanism that constrains the selection and order-of-inclusion of the components of a solution based on the value they are expected to provide.

5 Procedure

Algorithm 1 provides a pseudo-code listing of the Greedy Randomized Adaptive Search Procedure for minimizing a cost function.

Algorithm 1: Pseudo Code for the Greedy Randomized Adaptive Search Procedure.

```
   Input: α
   Output: S_best
1  S_best ← ConstructRandomSolution();
2  while ¬ StopCondition() do
3  |  S_candidate ← GreedyRandomizedConstruction(α);
4  |  S_candidate ← LocalSearch(S_candidate);
5  |  if Cost(S_candidate) < Cost(S_best) then
6  |  |  S_best ← S_candidate;
7  |  end
8  end
9  return S_best;
```

Algorithm 2 provides the pseudo-code the Greedy Randomized Construction function. The function involves the step-wise construction of a candidate solution using a stochastically greedy construction process. The functions works by building a Restricted Candidate List (RCL) that constraints the components of a solution (features) that may be selected from each cycle. The RCL may be constrained by an explicit size, or by using a threshold ($\alpha \in [0, 1]$) on the cost of adding each feature to the current candidate solution.

6 Heuristics

- The α threshold defines the amount of greediness of the construction mechanism, where values close to 0 may be too greedy, and values close to 1 may be too generalized.
- As an alternative to using the α threshold, the RCL can be constrained to the top $n\%$ of candidate features that may be selected from each construction cycle.
- The technique was designed for discrete problem classes such as combinatorial optimization problems.

7 Code Listing

Listing 1 provides an example of the Greedy Randomized Adaptive Search Procedure 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.

Algorithm 2: Pseudo Code Listing for the Greedy Randomized Construction function.

Input: α
Output: $S_{candidate}$

$S_{candidate} \leftarrow 0$;
while $S_{candidate} \neq$ ProblemSize **do**
  $Feature_{costs} \leftarrow 0$;
  for $Feature_i \notin S_{candidate}$ **do**
    $Feature_{costs} \leftarrow$ `CostOfAddingFeatureToSolution`($S_{candidate}$, $Feature_i$);
  end
  RCL $\leftarrow 0$;
  $Fcost_{min} \leftarrow$ `MinCost`($Feature_{costs}$);
  $Fcost_{max} \leftarrow$ `MaxCost`($Feature_{costs}$);
  for $F_icost \in Feature_{costs}$ **do**
    if $F_icost \leq Fcost_{min} + \alpha \cdot (Fcost_{max} - Fcost_{min})$ **then**
      RCL $\leftarrow Feature_i$;
    end
  end
  $S_{candidate} \leftarrow$ `SelectRandomFeature`(RCL);
end
return $S_{candidate}$;

The stochastic and greedy step-wise construction of a tour involves evaluating candidate cities by the the cost they contribute as being the next city in the tour. The algorithm uses a stochastic 2-opt procedure for the Local Search with a fixed number of non-improving iterations as the stopping condition.

```
MAX_ITERATIONS = 50
LOCAL_SEARCH_NO_IMPROVEMENTS = 100
GREEDINESS_FACTOR = 0.3
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 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
```

```ruby

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 construct_randomized_greedy_solution(cities, alpha)
  candidate = {}
  candidate[:vector] = [rand(cities.length)]
  allCities = Array.new(cities.length) {|i| i}
  while candidate[:vector].length < cities.length
    candidates = allCities - candidate[:vector]
    costs = Array.new(candidates.length){|i| euc_2d(cities[candidate[:vector].last], cities[i])}
    rcl, max, min = [], costs.max, costs.min
    costs.each_with_index { |c,i| rcl<<candidates[i] if c <= (min + alpha*(max-min)) }
    candidate[:vector] << rcl[rand(rcl.length)]
  end
  candidate[:cost] = cost(candidate[:vector], cities)
  return candidate
end

def search(cities, maxIterations, maxNoImprovementsLS, alpha)
  best = nil
  maxIterations.times do |iter|
    candidate = construct_randomized_greedy_solution(cities, alpha);
    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, GREEDINESS_FACTOR)
puts "Done. Best Solution: c=#{best[:cost]}, v=#{best[:vector].inspect}"
```

Listing 1: Greedy Randomized Adaptive Search Procedure algorithm in the Ruby Programming Language

8 References

8.1 Primary Sources

The seminal paper that introduces the general approach of stochastic and greedy step-wise construction of candidate solutions is by Feo and Resende [6]. The general approach was inspired by greedy heuristics by Hart and Shogan [11]. The seminal review paper that is cited with the preliminary paper is by Feo and Resende [4], and provides a coherent description of the GRASP technique, an example, and review of early applications. An early application was by Feo, Venkatraman and Bard for a machine scheduling problem [8]. Other early applications to scheduling problems include technical reports [5] (later published as [1]) and [7] (also later published as [9]).

8.2 Learn More

There are a vast number of review, application, and extension papers for GRASP. Pitsoulis and Resende provide an extensive contemporary overview of the field as a review chapter [13], as does Resende and Ribeiro that includes a clear presentation of the use of the α threshold parameter instead of a fixed size for the RCL. Festa and Resende provide an annotated bibliography as a review chapter that provides some needed insight into large amount of study that has gone into the approach [10]. There are numerous extensions to GRASP, not limited to the popular Reactive GRASP for adapting α [14], the use of long term memory to allow the technique to learn from candidate solutions discovered in previous iterations, and parallel implementations of the procedure such as 'Parallel GRASP' [12].

9 Conclusions

This report described the Greedy Randomized Adaptive Search Procedure for combinatorial optimization domains that uses an iterative process of stochastic and greedy step-wise construction of candidate solutions, followed by refinement with a Local Search procedure.

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] J.F. Bard, T.A. Feo, and S. Holland. A GRASP for scheduling printed wiring board assembly. *I.I.E. Trans.*, 28:155–165, 1996.

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

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

[4] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[5] T.A. Feo, J. Bard, and S. Holland. A grasp for scheduling printed wiring board assembly. Technical Report TX 78712-1063, Operations Research Group, Department of Mechanical Engineering, The University of Texas at Austin, 1993.

[6] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[7] T.A. Feo, K. Sarathy, and J. McGahan. A grasp for single machine scheduling with sequence dependent setup costs and linear delay penalties. Technical Report TX 78712-1063, Operations Research Group, Department of Mechanical Engineering, The University of Texas at Austin, 1994.

[8] T.A. Feo, K. Venkatraman, and J.F. Bard. A GRASP for a difficult single machine scheduling problem. *Computers & Operations Research*, 18:635–643, 1991.

[9] Thomas A. Feo, Kishore Sarathy, and John McGahan. A grasp for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers & Operations Research*, 23(9):881–895, 1996.

[10] P. Festa and M. G. C. Resende. *Essays and Surveys on Metaheuristics*, chapter GRASP: An annotated bibliography, pages 325–367. Kluwer Academic Publishers, 2002.

[11] J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.

[12] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel grasp implementation for the quadratic assignment problem. In *Parallel Algorithms for Irregularly Structured Problems (Irregular94)*, pages 111–130. Kluwer Academic Publishers, 1995.

[13] L. Pitsoulis and M. G. C. Resende. *Handbook of Applied Optimization*, chapter Greedy randomized adaptive search procedures, pages 168–181. Oxford University Press, 2002.

[14] M. Prais and C.C. Ribeiro. Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.