Reactive Tabu Search*

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

February 21, 2010
Technical Report: CA-TR-20100221-1

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 Reactive Tabu Search algorithm using the standardized template.

Keywords: `Clever, Algorithms, Description, Optimization, Reactive, Tabu, 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 [11]. 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 [13]. This report describes the Reactive Tabu Search algorithm using the standardized template.

2 Name

Reactive Tabu Search, RTS, R-TABU, Reactive Taboo Search

3 Taxonomy

Reactive Tabu Search is a Metaheuristic and a Global Optimization algorithm. It is an extension of Tabu Search [12] and the basis for a field of reactive techniques called Reactive Local Search and more broadly the field of Reactive Search Optimization.

4 Strategy

The objective of Tabu Search is to avoid cycles while applying a local search technique. The Reactive Tabu Search addresses this objective by explicitly monitoring the search and reacting to the occurrence of cycles and their repetition by adapting the tabu tenure (tabu list size). The strategy of the broader field of Reactive Search Optimization is to automate the process by which a practitioner configures a search procedure by monitoring its online behavior and to use machine learning techniques to adapt a techniques configuration.

5 Procedure

Algorithm 1 provides a pseudo-code listing of the Reactive Tabu Search algorithm for minimizing a cost function. The pseudo code is based on a the version of the Reactive Tabu Search described by Battiti and Tecchiolli in [8] with supplements like the `IsTabu` function from [6]. The procedure has been modified for brevity to exude the diversification procedure (escape move). Algorithm 2 describes the memory based reaction that manipulates the size of the `ProhibitionPeriod` in response to identified cycles in the ongoing search. Algorithm 3 describes the selection of the best move from a list of candidate moves in the neighborhood of a given solution. The function permits prohibited moves in the case where a prohibited move is better than the best know solution and the selected admissible move (called aspiration). Algorithm 4 determines whether a given neighborhood move is tabu based on the current `ProhibitionPeriod`, and is employed by sub-functions of the Algorithm 3 function.

Algorithm 1: Pseudo Code for the Reactive Tabu Search algorithm.

Input: $Iteration_{max}$, Increase, Decrease, ProblemSize
Output: S_{best}

S_{curr} ← `ConstructInitialSolution()`;
$S_{best} \leftarrow S_{curr}$;
TabuList ← 0;
ProhibitionPeriod ← 1;
foreach $Iteration_i \in Iteration_{max}$ **do**
    `MemoryBasedReaction(`Increase, Decrease, ProblemSize`)`;
    CandidateList ← `GenerateCandidateNeighborhood(`S_{curr}`)`;
    S_{curr} ← `BestMove(`CandidateList`)`;
    TabuList ← $Scurr_{feature}$;
    if `Cost(`S_{curr}`)` $\leq$ `Cost(`S_{best}`)` **then**
        $S_{best} \leftarrow S_{curr}$;
    end
end
return S_{best};

6 Heuristics

- Reactive Tabu Search is an extension of Tabu Search and as such should exploit the best practices used for the parent algorithm.
- Reactive Tabu Search was designed for discrete domains such as combinatorial optimization, although has been applied to continuos function optimization.
- Reactive Tabu Search was proposed to use efficient memory data structures such as hash tables.

Algorithm 2: Pseudo Code for the `MemoryBasedReaction` function in the Reactive Tabu Search algorithm.

Input: Increase, Decrease, ProblemSize
Output:
if `HaveVisitedSolutionBefore`(S_{curr}, VisitedSolutions) **then**
    $Scurr_t \leftarrow$ `RetrieveLastTimeVisited`(VisitedSolutions, S_{curr});
    RepetitionInterval $\leftarrow Iteration_i - Scurr_t$;
    $Scurr_t \leftarrow Iteration_i$;
    if RepetitionInterval $< 2 *$ ProblemSize **then**
        $RepetitionInterval_{avg} \leftarrow 0.1 *$ RepetitionInterval $+ 0.9 * RepetitionInterval_{avg}$;
        ProhibitionPeriod $\leftarrow$ ProhibitionPeriod $*$ Increase;
        $ProhibitionPeriod_t \leftarrow Iteration_i$;
    end
else
    VisitedSolutions $\leftarrow S_{curr}$;
    $Scurr_t \leftarrow Iteration_i$;
end
if $Iteration_i$-$ProhibitionPeriod_t > RepetitionInterval_{avg}$ **then**
    ProhibitionPeriod $\leftarrow$ `Max`(*1*, ProhibitionPeriod $*$ Decrease);
    $ProhibitionPeriod_t \leftarrow Iteration_i$;
end

Algorithm 3: Pseudo Code for the `BestMove` function in the Reactive Tabu Search algorithm.

Input: ProblemSize
Output: S_{curr}
$CandidateList_{admissible} \leftarrow$ `GetAdmissibleMoves`(CandidateList);
$CandidateList_{tabu} \leftarrow$ CandidateList $- CandidateList_{admissible}$;
if `Size`($CandidateList_{admissible}$) < 2 **then**
    ProhibitionPeriod $\leftarrow$ ProblemSize $- 2$;
    $ProhibitionPeriod_t \leftarrow Iteration_i$;
end
$S_{curr} \leftarrow$ `GetBest`($CandidateList_{admissible}$);
$Sbest_{tabu} \leftarrow$ `GetBest`($CandidateList_{tabu}$);
if `Cost`($Sbest_{tabu}$) $<$ `Cost`(S_{best}) $\wedge$ `Cost`($Sbest_{tabu}$) $<$ `Cost`(S_{curr}) **then**
    $S_{curr} \leftarrow Sbest_{tabu}$;
end
return S_{curr};

Algorithm 4: Pseudo Code for the `IsTabu` function in the Reactive Tabu Search algorithm.

Input:
Output: Tabu
Tabu $\leftarrow$ FALSE;
$Scurr^t_{feature} \leftarrow$ `RetrieveTimeFeatureLastUsed`($Scurr_{feature}$);
if $Scurr^t_{feature} \geq Iteration_{curr}$ $-$ProhibitionPeriod **then**
    Tabu $\leftarrow$ TRUE;
end
return Tabu;

- Reactive Tabu Search was proposed to use an long-term memory to diversify the search after a threshold of cycle repetitions has been reached.
- The `increase` parameter should be greater than one (such as 1.1 or 1.3) and the `decrease` parameter should be less than one (such as 0.9 or 0.8).

7 Code Listing

Listing 1 provides an example of the Reactive Tabu 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 procedure is based on the code listing described by Battiti and Tecchiolli in [8] with supplements like the `IsTabu` function from [6]. The implementation does not use efficient memory data structures such as hash tables. The algorithm is initialized with a stochastic 2-opt local search, and the neighborhood is generated as a fixed candidate list of stochastic 2-opt moves. The edges selected for changing in the 2-opt move are stored as features in the tabu list. The example does not implement the escape procedure for search diversification.

```
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, [[permutation[c1-1], permutation[c1]], [permutation[c2-1], permutation[c2]]]
end

def generate_initial_solution(cities, maxNoImprovements)
  best = {}
  best[:vector] = random_permutation(cities)
  best[:cost] = cost(best[:vector], cities)
  noImprovements = 0
  begin
    candidate = {}
    candidate[:vector] = stochastic_two_opt(best[:vector])[0]
    candidate[:cost] = cost(candidate[:vector], cities)
    if candidate[:cost] <= best[:cost]
      noImprovements, best = 0, candidate
    else
      noImprovements += 1
```

```ruby
    end
  end until noImprovements >= maxNoImprovements
  return best
end

def is_tabu?(edge, tabuList, iteration, prohibitionPeriod)
  tabuList.each do |entry|
    if entry[:edge] == edge
      if entry[:iteration] >= iteration-prohibitionPeriod
        return true
      else
        return false
      end
    end
  end
  return false
end

def make_tabu(tabuList, edge, iteration)
  tabuList.each do |entry|
    if entry[:edge] == edge
      entry[:iteration] = iteration
      return entry
    end
  end
  entry = {}
  entry[:edge] = edge
  entry[:iteration] = iteration
  tabuList.push(entry)
  return entry
end

def to_edge_list(permutation)
  list = []
  permutation.each_with_index do |c1, i|
    c2 = (i==permutation.length-1) ? permutation[0] : permutation[i+1]
    c1, c2 = c2, c1 if c1 > c2
    list << [c1, c2]
  end
  return list
end

def equivalent_permutations(edgelist1, edgelist2)
  edgelist1.each do |edge|
    return false if !edgelist2.include?(edge)
  end
  return true
end

def generate_candidate(best, cities)
  candidate = {}
  candidate[:vector], edges = stochastic_two_opt(best[:vector])
  candidate[:cost] = cost(candidate[:vector], cities)
  return candidate, edges
end

def get_candidate_entry(visitedList, permutation)
  edgeList = to_edge_list(permutation)
  visitedList.each do |entry|
    return entry if equivalent_permutations(edgeList, entry[:edgelist])
  end
  return nil
end
```

```ruby

def store_permutation(visitedList, permutation, iteration)
  entry = {}
  entry[:edgelist] = to_edge_list(permutation)
  entry[:iteration] = iteration
  entry[:visits] = 1
  visitedList.push(entry)
  return entry
end

def sort_neighbourhood(candidates, tabuList, prohibitionPeriod, iteration)
  tabu, admissable = [], []
  candidates.each do |a|
    if is_tabu?(a[1][0], tabuList, iteration, prohibitionPeriod) or
      is_tabu?(a[1][1], tabuList, iteration, prohibitionPeriod)
      tabu << a
    else
      admissable << a
    end
  end
  return tabu, admissable
end

def search(cities, maxNoImprove, candidateListSize, maxIterations, increase, decrease)
  current = generate_initial_solution(cities, maxNoImprove)
  best = current
  tabuList, prohibitionPeriod = [], 1
  visitedList, avgLength, lastChange = [], 1, 0
  maxIterations.times do |iter|
    candidateEntry = get_candidate_entry(visitedList, current[:vector])
    if !candidateEntry.nil?
      repetitionInterval = iter - candidateEntry[:iteration]
      candidateEntry[:iteration] = iter
      candidateEntry[:visits] += 1
      if repetitionInterval < 2*(cities.length-1)
        avgLength = 0.1*(iter-candidateEntry[:iteration]) + 0.9*avgLength
        prohibitionPeriod = (prohibitionPeriod.to_f * increase)
        lastChange = iter
      end
    else
      store_permutation(visitedList, current[:vector], iter)
    end
    if iter-lastChange > avgLength
      prohibitionPeriod = [prohibitionPeriod*decrease,1].max
      lastChange = iter
    end
    candidates = Array.new(candidateListSize) {|i| generate_candidate(current, cities)}
    candidates.sort! {|x,y| x.first[:cost] <=> y.first[:cost]}
    tabu, admissible = sort_neighbourhood(candidates, tabuList, prohibitionPeriod, iter)
    if admissible.length < 2
      prohibitionPeriod = cities.length-2
      lastChange = iter
    end
    current, bestMoveEdges = admissible.first if !admissible.empty?
    if !tabu.empty? and tabu.first[0][:cost]<best[:cost] and tabu.first[0][:cost]<current[:cost]
      current, bestMoveEdges = tabu.first
    end
    bestMoveEdges.each {|edge| make_tabu(tabuList, edge, iter)}
    best = candidates.first[0] if candidates.first[0][:cost] < best[:cost]
    puts " > iteration #{(iter+1)}, tabuList=#{tabuList.length},
        prohibitionPeriod=#{prohibitionPeriod.round}, best: c=#{best[:cost]}"
  end
  return best
```

```ruby
end

MAX_ITERATIONS = 300
MAX_NO_IMPROVE = 50
MAX_CANDIDATES = 50
INCREASE = 1.3
DECREASE = 0.9
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]]

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

Listing 1: Reactive Tabu Search algorithm in the Ruby Programming Language

8 References

8.1 Primary Sources

Reactive Tabu Search was proposed by Battiti and Tecchiolli as an extension to Tabu Search that included an adaptive tabu list size in addition to a diversification mechanism [6]. The technique also used efficient memory structures that were based on an earlier work by Battiti and Tecchiolli that considered a parallel tabu search [5]. Some early application papers by Battiti and Tecchiolli include a comparison to Simulated Annealing applied to the Quadratic Assignment Problem [7], benchmarked on instances of the knapsack problem and N-K models and compared with Repeated Local Minima Search, Simulated Annealing, and Genetic Algorithms [8], and training neural networks on an array of problem instances [9].

8.2 Learn More

Reactive Tabu Search was abstracted to a form called Reactive Local Search that considers adaptive methods that learn suitable parameters for heuristics that manage an embedded local search technique [3, 4]. Under this abstraction, the Reactive Tabu Search algorithm is single example of the Reactive Local Search principle applied to the Tabu Search. This framework was further extended to the use of any adaptive machine learning techniques to adapt the parameters of an algorithm by reacting to algorithm outcomes online while solving a problem, called Reactive Search [10]. The best reference for this general framework is the book on Reactive Search Optimization by Battiti, Brunato, and Mascia [2]. Additionally, the review chapter by Battiti and Brunato provides a contemporary description [1].

9 Conclusions

This report described the Reactive Tabu search as an extension of Tabu Search that adapts the tabu tenure based on feedback from the problem instance. This algorithm was particularly difficult to describe given the large amount detailed pseudo code examples used in the literature to present the approach and the general complexity of the algorithm.

A feature identified in the preparation of this report that may be a useful contribution to the Clever Algorithms project was the description of pseudo code elements before the presentation of pseudo code in the Reactive Tabu Search literature. A description of the pseudo code idioms (condition constructs, flow constructs, assuagement, etc) should be described in introductory material for all algorithm descriptions, such as an introduction chapter in the book. Additionally, the terms used on each algorithm procedure should be clearly defined to avoid ambiguity.

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] R. Battiti and M. Brunato. *Handbook of Metaheuristics*, chapter Reactive Search Optimization: Learning while Optimizing. Springer Verlag, 2nd edition, 2009.

[2] R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*. Springer, 2008.

[3] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. Technical Report TR-95-052, International Computer Science Institute, Berkeley, CA, 1995.

[4] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4):610–637, 2001.

[5] R. Battiti and G. Tecchiolli. Parallel biased search for combinatorial optimization: genetic algorithms and tabu. *Microprocessors and Microsystems*, 16(7):351–367, 1992.

[6] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[7] R. Battiti and G. Tecchiolli. Simulated annealing and tabu search in the long run: a comparison on qap tasks. *Computer and Mathematics with Applications*, 28(6):1–8, 1994.

[8] R. Battiti and G. Tecchiolli. Local search with memory: Benchmarking rts. *Operations Research Spektrum*, 17(2/3):67–86, 1995.

[9] R. Battiti and G. Tecchiolli. Training neural nets with the reactive tabu search. *IEEE Transactions on Neural Networks*, 6(5):1185–1200, 1995.

[10] Roberto Battiti. Machine learning methods for parameter tuning in heuristics. In *5th DIMACS Challenge Workshop: Experimental Methodology Day*, 1996.

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

[12] Jason Brownlee. Tabu search. Technical Report CA-TR-20100213-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, February 2010.

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