Tabu Search*

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

February 13, 2010
Technical Report: CA-TR-20100213-1

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 Tabu Search algorithm using the standardized template.

Keywords: `Clever, Algorithms, Description, Optimization, 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 [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 Tabu Search algorithm using the standardized template.

2 Name

Tabu Search, TS, Taboo Search

3 Taxonomy

Tabu Search is a Global Optimization algorithm and a Metaheuristic or Meta-strategy for controlling an embedded heuristic technique. Tabu Search is a parent for a large family of derivative approaches that introduce memory structures in Metaheuristics, such as Reactive Tabu Search and Parallel Tabu Search.

4 Strategy

The objective for the Tabu Search algorithm is to constrain an embedded heuristic from returning to recently visited areas of the search space, referred to as cycling. The strategy of the approach is to maintain a short term memory of the specific changes of recent moves within the search space and preventing future moves from undoing those changes. Additional intermediate-term memory structures may be introduced to bias moves toward promising areas of the search space, as well as longer-term memory structures that promote a general diversity in the search across the search space.

5 Procedure

Algorithm 1 provides a pseudo-code listing of the Tabu Search algorithm for minimizing a cost function. The listing shows the simple Tabu Search algorithm with short term memory, without intermediate and long term memory management.

Algorithm 1: Pseudo Code for the Tabu Search algorithm.

```
Input: TabuList_size
Output: S_best
S_best ← ConstructInitialSolution();
TabuList ← 0;
while ¬ StopCondition() do
    CandidateList ← 0;
    for S_candidate ∈ Sbest_neighborhood do
        if ¬ ContainsAnyFeatures(S_candidate, TabuList) then
            CandidateList ← S_candidate;
        end
    end
    S_candidate ← LocateBestCandidate(CandidateList);
    if Cost(S_candidate) ≤ Cost(S_best) then
        S_best ← S_candidate;
        TabuList ← FeatureDifferences(S_candidate, S_best);
        while TabuList > TabuList_size do
            DeleteFeature(TabuList);
        end
    end
end
return S_best;
```

6 Heuristics

- Tabu search was designed to manage an embedded hill climbing heuristic, although may be adapted to manage any neighborhood exploration heuristic.
- Tabu search was designed for, and has predominately been applied to discrete domains such as combinatorial optimization problems.
- Candidates for neighboring moves can be generated deterministically for the entire neighborhood or the neighborhood can be stochastically sampled to a fixed size, trading off efficiency for accuracy.

- Intermediate-term memory structures can be introduced (complementing the short-term memory) to focus the search on promising areas of the search space (intensification), called aspiration criteria.

- Long-term memory structures can be introduced (complementing the short-term memory) to encourage useful exploration of the broader search space, called diversification. Strategies may include generating solutions with rarely used components and biasing generation away from the most commonly used solution components.

7 Code Listing

Listing 1 provides an example of the 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 algorithm is an implementation of the simple Tabu Search with a short term memory structure that executes for a fixed number of iterations. The starting point for the search is prepared using a random permutation that is refined using a stochastic 2-opt Local Search procedure. The stochastic 2-opt procedure is used as the embedded hill climbing heuristic with a fixed sized candidate list. The two edges that are deleted in each 2-opt move are stored on the tabu list. This general approach is similar to that used by Knox in his work on Tabu Search for symmetrical TSP [14] and Fiechter for the Parallel Tabu Search for the TSP [4].

```
MAX_ITERATIONS = 100
MAX_NO_IMPROVEMENTS = 50
TABU_LIST_SIZE = 15
MAX_CANDIDATES = 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
```

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

def is_tabu?(permutation, tabuList)
  permutation.each_with_index do |c1, i|
    c2 = (i==permutation.length-1) ? permutation[0] : permutation[i+1]
    tabuList.each do |forbidden_edge|
      return true if forbidden_edge == [c1, c2]
    end
  end
  return false
end

def generate_candidate(best, tabuList, cities)
  permutation, edges = nil, nil
  begin
    permutation, edges = stochastic_two_opt(best[:vector])
  end while is_tabu?(permutation, tabuList)
  candidate = {}
  candidate[:vector] = permutation
  candidate[:cost] = cost(candidate[:vector], cities)
  return candidate, edges
end

def search(cities, tabuListSize, candidateListSize, maxIterations, maxNoImprovementsLS)
  best = generate_initial_solution(cities, maxNoImprovementsLS)
  tabuList = Array.new(tabuListSize)
  maxIterations.times do |iter|
    candidates = Array.new(candidateListSize) {|i| generate_candidate(best, tabuList, cities)}
    candidates.sort! {|x,y| x.first[:cost] <=> y.first[:cost]}
    bestCandidate = candidates.first[0]
    bestCandidateEdges = candidates.first[1]
    if(bestCandidate[:cost] < best[:cost])
      best = bestCandidate
      bestCandidateEdges.each do |edge|
        tabuList.pop
        tabuList.push(edge)
      end
    end
    puts " > iteration #{(iter+1)}, best: c=#{best[:cost]}"
  end
  return best
end

best = search(BERLIN52, TABU_LIST_SIZE, MAX_CANDIDATES, MAX_ITERATIONS, MAX_NO_IMPROVEMENTS)
```

```ruby
puts "Done. Best Solution: c=#{best[:cost]}, v=#{best[:vector].inspect}"
```

Listing 1: Tabu Search algorithm in the Ruby Programming Language

8 References

8.1 Primary Sources

Tabu Search was introduced by Glover applied to scheduling employees to duty rosters [11] and a more general overview in the context of the TSP [7], based on his previous work on surrogate constraints on integer programming problems [6]. Glover provided a seminal overview of the algorithm in a two-part journal article, the first part of which introduced the algorithm, and reviewed then-recent applications [8], and the second which focused on advanced topics and open areas of research [9].

8.2 Learn More

Glover provides a high-level introduction to Tabu Search in the form of a practical tutorial [10], as does Glover and Taillard in a user guide format [12]. The best source of information for Tabu Search is the book dedicated to the approach by Glover and Laguna that covers the principles of the technique in detail as well as an in-depth review of applications [13]. The approach appeared in Science, that considered a modification for its application to continuous function optimization problems [3]. Finally, Gendreau provides an excellent contemporary review of the algorithm, highlighting best practices and application heuristics collected from across the field of study [5].

9 Conclusions

This report described the Tabu Search algorithm as an approach that adds a memory structure to an existing hill climbing heuristic to prohibit cycling-like behavior while navigating the search space.

Some research discovered during the preparation of this report that may prove useful to the Clever Algorithms project was a list of heuristics for applying a previously unknown technique to a given problem by Gendreau [5]. These heuristics may be generalized and provided and elaborated as an advanced topic to accompany with algorithm descriptions prepared for the project.

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] D. Cvijovic and J. Klinowski. Taboo search: An approach to the multiple minima problem. *Science*, 267:664–666, 1995.

[4] C.-N. Fiechter. A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics*, 3(6):243–267, 1994.

[5] Michel Gendreau. *Handbook of Metaheuristics*, chapter 2: An Introduction to Tabu Search, pages 37–54. Springer, 2003.

[6] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[7] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[8] F. Glover. Tabu search – Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[9] F Glover. Tabu search – Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[10] F. Glover. Tabu search: A tutorial. *Interfaces*, 4:74–94, 1990.

[11] F. Glover and C. McMillan. The general employee scheduling problem: an integration of ms and ai. *Computers and Operations Research*, 13(5):536–573, 1986.

[12] Fred Glover and Eric Taillard. A user's guide to tabu search. *Annals of Operations Research*, 41(1):1–28, 1993.

[13] Fred W. Glover and Manuel Laguna. *Tabu Search*. Springer, 1998.

[14] John Knox. Tabu search performance on the symmetric traveling salesman problem. *Computers & Operations Research*, 21(8):867–876, 1994.