Ant Colony System*

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

November 15, 2010
Technical Report: CA-TR-20101115a-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 Ant Colony System algorithm.

Keywords: `Clever, Algorithms, Description, Optimization, Ant, Colony, System`

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 Ant Colony System algorithm.

2 Name

Ant Colony System, ACS, Ant-Q

3 Taxonomy

The Ant Colony System algorithm is an example of an Ant Colony Optimization method from the field of Swarm Intelligence, Metaheuristics and Computational Intelligence. Ant Colony System is an extension to the Ant System algorithm and is related to other Ant Colony Optimization methods such as Elite Ant System, Rank-based Ant System, and Ant Colony System.

*© Copyright 2010 Jason Brownlee. Some Rights Reserved. This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 2.5 Australia License.

4 Inspiration

The Ant Colony System algorithm is inspired by the foraging behavior of ants, specifically the pheromone communication between ants regarding a good path between the colony and a food source in an environment. This mechanism is called stigmergy.

5 Metaphor

Ants initially wander randomly around their environment. Once food is located an ant will begin laying down pheromone in the environment. Numerous trips between the food and the colony are performed and if the same route is followed that leads to food then additional pheromone is laid down. Pheromone decays in the environment, so that older paths are less likely to be followed. Other ants may discover the same path to the food and in turn may follow it and also lay down pheromone. A positive feedback process routes more and more ants to productive paths that are in turn further refined through use.

6 Strategy

The objective of the strategy is to exploit historic and heuristic information to construct candidate solutions and fold the information learned from constructing solutions into the history. Solutions are constructed one discrete piece at a time in a probabilistic step-wise manner. The probability of selecting a component is determined by the heuristic contribution of the component to the overall cost of the solution and the quality of solutions from which the component has historically known to have been included. History is updated proportional to the quality of candidate solutions and is uniformly deceased ensuring the most recent and useful information is retained.

7 Procedure

Algorithm 1 provides a pseudo-code listing of the main Ant Colony System algorithm for minimizing a cost function. The probabilistic step-wise construction of solution makes use of both history (pheromone) and problem-specific heuristic information to incrementally construction a solution piece-by-piece. Each component can only be selected if it has not already been chosen (for most combinatorial problems), and for those components that can be selected from given the current component i, their probability for selection is defined as:

$$P_{i,j} \leftarrow \frac{\tau_{i,j}^{\alpha} \times \eta_{i,j}^{\beta}}{\sum_{k=1}^{c} \tau_{i,k}^{\alpha} \times \eta_{i,k}^{\beta}} \tag{1}$$

where $\tau_{i,j}$ is the maximizing contribution to the overall score of selecting the component (such as $\frac{1.0}{distance_{i,j}}$ for the Traveling Salesman Problem), α is the heuristic coefficient (commonly fixed at 1.0), $\eta_{i,j}$ is the pheromone value for the component, β is the history coefficient, and c is the set of usable components. A greediness factor ($q0$) is used to influences when to use the above probabilistic component selection and when to greedily select the best possible component.

A local pheromone update is performed for each solution that is constructed to dissuade following solutions to use the same components in the same order, as follows:

$$\tau_{i,j} \leftarrow (1 - \sigma) \times \tau_{i,j} + \sigma \times \tau_{i,j}^{0} \tag{2}$$

where $\tau_{i,j}$ represents the pheromone for the component (graph edge) (i, j), σ is the local pheromone factor, and $\tau_{i,j}^{0}$ is the initial pheromone value.

At the end of each iteration, the pheromone is updated and decayed using the best candidate solution found thus far (or the best candidate solution found for the iteration), as follows:

$$\tau_{i,j} \leftarrow (1-\rho) \times \tau_{i,j} + \Delta_{i,j}^{best} \tag{3}$$

where $\tau_{i,j}$ represents the pheromone for the component (graph edge) (i,j), ρ is the decay factor, and $\Delta_{i,j}^{best}$ is the maximizing solution cost for the best solution found so far.

Algorithm 1: Pseudo Code for the Ant Colony System algorithm.

```
Input: ProblemSize, $Population_{size}$, $m$, $\rho$, $\beta$, $\sigma$, $q0$
Output: $P_{best}$
1  $P_{best} \leftarrow$ CreateHeuristicSolution(ProblemSize);
2  $Pbest_{cost} \leftarrow$ Cost($S_h$);
3  $Pheromone_{init} \leftarrow \frac{1.0}{\text{ProblemSize} \times Pbest_{cost}}$;
4  Pheromone $\leftarrow$ InitializePheromone($Pheromone_{init}$);
5  while $\neg$StopCondition() do
6      for $i = 1$ to $m$ do
7          $S_i \leftarrow$ ConstructSolution(Pheromone, ProblemSize, $\beta$, $q0$);
8          $Si_{cost} \leftarrow$ Cost($S_i$);
9          if $Si_{cost} \leq Pbest_{cost}$ then
10             $Pbest_{cost} \leftarrow Si_{cost}$;
11             $P_{best} \leftarrow S_i$;
12         end
13         LocalUpdateAndDecayPheromone(Pheromone, $S_i$, $Si_{cost}$, $\sigma$);
14     end
15     GlobalUpdateAndDecayPheromone(Pheromone, $P_{best}$, $Pbest_{cost}$, $\rho$);
16 end
17 return $P_{best}$;
```

8 Heuristics

- The Ant Colony System algorithm was designed for use with combinatorial problems such as the TSP, knapsack problem, quadratic assignment problems, graph coloring problems and many others.
- The local pheromone (history) coefficient (σ) controls the amount of contribution history plays in a components probability of selection and is commonly set to 0.1.
- The heuristic coefficient (β) controls the amount of contribution problem-specific heuristic information plays in a components probability of selection and is commonly between 2 and 5, such as 2.5.
- The decay factor (ρ) controls the rate at which historic information is lost and is commonly set to 0.1.
- The greediness factor ($q0$) is commonly set to 0.9.
- The total number of ants (m) is commonly set low, such as 10.

9 Code Listing

Listing 1 provides an example of the Ant Colony System 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. Some extensions to the algorithm implementation for speed improvements may consider pre-calculating a distance matrix for all the cities in the problem, and pre-computing a probability matrix for choices during the probabilistic step-wise construction of tours.

```ruby
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 initialise_pheromone_matrix(num_cities, init_pher)
  return Array.new(num_cities){|i| Array.new(num_cities, init_pher)}
end

def nearest_neighbor_solution(cities)
  candidate = {}
  candidate[:vector] = [rand(cities.length)]
  all_cities = Array.new(cities.length) {|i| i}
  while candidate[:vector].length < cities.length
    next_city = {:city=>nil,:dist=>nil}
    candidates = all_cities - candidate[:vector]
    candidates.each do |city|
      dist = euc_2d(cities[candidate[:vector].last], city)
      if next_city[:city].nil? or next_city[:dist] < dist
        next_city[:city] = city
        next_city[:dist] = dist
      end
    end
    candidate[:vector] << next_city[:city]
  end
  candidate[:cost] = cost(candidate[:vector], cities)
  return candidate
end

def calculate_choices(cities, last_city, exclude, pheromone, c_heuristic, c_history)
  choices = []
  cities.each_with_index do |coord, i|
    next if exclude.include?(i)
    prob = {:city=>i}
    prob[:history] = pheromone[last_city][i] ** c_history
    prob[:distance] = euc_2d(cities[last_city], coord)
    prob[:heuristic] = (1.0/prob[:distance]) ** c_heuristic
    prob[:prob] = prob[:history] * prob[:heuristic]
    choices << prob
  end
  choices
end

```

```ruby
def prob_select_next_city(choices)
  sum = choices.inject(0.0){|sum,element| sum + element[:prob]}
  return choices[rand(choices.length)][:city] if sum == 0.0
  v, next_city = rand(), -1
  choices.each_with_index do |choice, i|
    if i==choices.length-1
      next_city = choice[:city]
    else
      v -= (choice[:prob]/sum)
      if v <= 0.0
        next_city = choice[:city]
        break
      end
    end
  end
  return next_city
end

def greedy_select_next_city(choices)
  best = choices.max{|a,b| a[:prob]<=>b[:prob]}
  return best[:city]
end

def stepwise_construction(cities, pheromone, c_heuristic, c_greediness)
  perm = []
  perm << rand(cities.length)
  begin
    choices = calculate_choices(cities, perm.last, perm, pheromone, c_heuristic, 1.0)
    greedy = rand() <= c_greediness
    next_city = (greedy) ? greedy_select_next_city(choices) : prob_select_next_city(choices)
    perm << next_city
  end until perm.length == cities.length
  return perm
end

def global_update_pheromone(pheromone, candidate, decay_factor)
  candidate[:vector].each_with_index do |x, i|
    y = (i==candidate[:vector].length-1) ? candidate[:vector][0] : candidate[:vector][i+1]
    value = ((1.0-decay_factor)*pheromone[x][y]) + (decay_factor*(1.0/candidate[:cost]))
    pheromone[x][y] = value
    pheromone[y][x] = value
  end
end

def local_update_pheromone(pheromone, candidate, c_local_pheromone, init_pheromone)
  candidate[:vector].each_with_index do |x, i|
    y = (i==candidate[:vector].length-1) ? candidate[:vector][0] : candidate[:vector][i+1]
    value = ((1.0-c_local_pheromone)*pheromone[x][y]) + (c_local_pheromone * init_pheromone)
    pheromone[x][y] = value
    pheromone[y][x] = value
  end
end

def search(cities, max_iterations, num_ants, decay_factor, c_heuristic, c_local_pheromone,
    c_greediness)
  best = nearest_neighbor_solution(cities)
  init_pheromone = 1.0 / (cities.length.to_f * best[:cost])
  puts "Nearest Neighbor heuristic solution: cost=#{best[:cost]}"
  pheromone = initialise_pheromone_matrix(cities.length, init_pheromone)
  max_iterations.times do |iter|
    solutions = []
    num_ants.times do
      candidate = {}
      candidate[:vector] = stepwise_construction(cities, pheromone, c_heuristic, c_greediness)
      candidate[:cost] = cost(candidate[:vector], cities)
      best = candidate if candidate[:cost] < best[:cost]
      local_update_pheromone(pheromone, candidate, c_local_pheromone, init_pheromone)
    end
    global_update_pheromone(pheromone, best, decay_factor)
    puts " > iteration #{(iter+1)}, best=#{best[:cost]}"
  end
  return best
end

if __FILE__ == $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]]
  max_iterations = 100
  num_ants = 10
  decay_factor = 0.1
  c_heuristic = 2.5
  c_local_pheromone = 0.1
  c_greediness = 0.9

  best = search(berlin52, max_iterations, num_ants, decay_factor, c_heuristic,
      c_local_pheromone, c_greediness)
  puts "Done. Best Solution: c=#{best[:cost]}, v=#{best[:vector].inspect}"
end
```

Listing 1: Ant Colony System algorithm in the Ruby Programming Language

10 References

10.1 Primary Sources

The algorithm was initially investigated by Dorigo and Gambardella under the name Ant-Q [6, 8]. It was renamed Ant Colony System and further investigated first in a technical report by Dorigo and Gambardella [5], and later published [4].

10.2 Learn More

The seminal book on Ant Colony Optimization in general with a detailed treatment of Ant Colony System is "Ant colony optimization" by Dorigo and Stützle [7]. An earlier book "Swarm intelligence: from natural to artificial systems" by Bonabeau, Dorigo, and Theraulaz also provides an introduction to Swarm Intelligence with a detailed treatment of Ant Colony System [1].

11 Conclusions

This report described the Ant Colony System algorithm using the standardized algorithm template.

12 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 subjected to continuous improvement. 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] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems.* Oxford University Press US, 1999.

[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] M. Dorigo and L. M. Gambardella. Ant colony system : A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[5] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problems. Technical Report TR/IRIDIA/1996-5, IRIDIA, Universit Libre de Bruxelles, 1997.

[6] M. Dorigo and L.M. Gambardella. A study of some properties of ant-q. In I. Rechenberg H.M. Voigt, W. Ebeling and H.S. Schwefel, editors, *Proceedings of PPSN IVFourth International Conference on Parallel Problem Solving From Nature*, pages 656–665. Springer-Verlag, 1996.

[7] Marco Dorigo and Thomas Stützle. *Ant colony optimization.* MIT Press, 2004.

[8] L. Gambardella and M. Dorigo. Ant-q: a reinforcement learning approach to the traveling salesman problem. In A. Prieditis and S. Russell, editors, *Proceedings of ML-95, Twelfth International Conference on Machine Learning*, pages 252–260. Morgan Kaufmann, 1995.