International Journal of Computational Intelligence and Applications

The Accumulated Experience Ant Colony for the Traveling Salesman Problem

JAMES MONTGOMERY*
MARCUS RANDALL
School of Information Technology, Bond University
QLD 4229, Australia

Ant colony optimization techniques are usually guided by pheromone and heuristic cost information when choosing the next element to add to a solution. However, while an individual element may be attractive, usually its long term consequences are neither known nor considered. For instance, a short link in a traveling salesman problem may be incorporated into an ant's solution, yet, as a consequence of this link, the rest of the path may be longer than if another link was chosen. The Accumulated Experience Ant Colony uses the previous experiences of the colony to guide in the choice of elements. This is in addition to the normal pheromone and heuristic costs. Two versions of the algorithm are presented, the original and an improved AEAC that makes greater use of accumulated experience. The results indicate that the original algorithm finds improved solutions on problems with less than 100 cities, while the improved algorithm finds better solutions on larger problems.

Keywords: Ant colony optimization; traveling salesman problem.

1. Introduction

Ant Colony Optimization (ACO) meta-heuristics have proved to be remarkably successful in solving a range of discrete optimization problems. The core principal on which these techniques operate is to use a collective memory of the characteristics of the problem being solved, built up over time. Each ant consults this memory when augmenting its solution. This memory usually stores the colony's preference for adding a particular element with respect to the current element. For instance, given a traveling salesman problem (TSP) in which a Hamiltonian circuit of minimum total length is sought, an ant would choose the next city based on the pheromone amount associated between the current city and the potential next city in conjunction with the distance. The accumulated experience ant colony (AEAC) adds another dimension to this by considering how the choice of elements affects the solution quality *after* it has been incorporated. As such, we have modified the characteristic element selection equations to incorporate a weighting term for the

*This author is a PhD scholar supported by an Australian Postgraduate Award.

accumulated experience component. This weighting is based on the characteristics of partial solutions generated within the current iteration. Elements that appear to lead to better solutions are valued more highly, while those that lead to poorer solutions are made less desirable. Thus, we augment the usual autocatalytic behavior of ACO by providing more immediate and objective feedback on the quality of the choices made. In essence, AEAC is less greedy than standard ACO techniques because it considers the long-term consequences of elements.

Two versions of the algorithm are presented. The first is the AEAC in its original form, while the second is a modified algorithm that uses element weightings generated in previous iterations as well as the current one so that experience can be accumulated for a greater number of elements.

Heusse, Guérin, Snyders and Kuntz propose a similar information sharing system for routing in packet-switched networks, called Co-operating Asymmetric Forward routing (CAF routing).[1] Nodes in CAF routing send out agents to calculate the current delay on network routes. The exact route each agent takes is based on the same routing information used for normal data packets, but is modified by information gathered by agents traveling on the same route in the opposite direction. Thus, CAF routing uses knowledge gathered from many agents to report on the current state of network routes. This differs from AEAC, however, which uses accumulated experience in a heuristic fashion to *estimate* the utility of individual solution elements.

This paper is organized as follows. Section 2 has a brief overview of ACO while Section 3 explains how we adapt it to incorporate the accumulated experience component. Section 4 describes the development of an improved AEAC. Section 5 shows the results of the two AEACs on some benchmark TSPs while Section 6 gives the conclusions of this work.

2. ACO

ACO is an umbrella term for a number of similar meta-heuristics.[2] The Ant Colony System (ACS) meta-heuristic applied to the TSP will be described here in order to demonstrate the underlying principles of ACO.[3]

Consider a set of cities, with known distances between each pair of cities. The aim of the TSP is to find the shortest path to traverse all cities exactly once and return to the starting city. ACS is applied to this problem in the following way. Consider a TSP with N cities. Cities i and j are separated by distance $d(i, j)$. Scatter m ants randomly on these cities ($m \leq N$). In discrete time steps, all ants select their next city then simultaneously move to their next city. Ants deposit a substance known as *pheromone* to communicate with the colony about the utility (goodness) of the edges. Denote the accumulated strength of pheromone on edge (i, j) by $\tau(i, j)$.

At the commencement of each time step, Eqs. (1) and (2) are used to select the next city s for ant k currently at city r. Eq. (1) is a greedy selection tech-

nique favoring cities which possess the best combination of short distance and large pheromone levels. Eq. (2) balances this by allowing a probabilistic selection of the next city.

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \left\{ \tau(r,u)[d(r,u)]^{\beta} \right\} & \text{if } q \leq q_0 \\ \text{Eq. (2)} & \text{otherwise.} \end{cases} \tag{1}$$

$$p_k(r,s) = \begin{cases} \frac{\tau(r,s)[d(r,s)]^{\beta}}{\sum_{u \in J_k(r)} \tau(r,u)[d(r,u)]^{\beta}} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Note that $q \in [0,1]$ is a uniform random number and q_0 is a parameter. To maintain the restriction of unique visitation, ant k is prohibited from selecting a city which it has already visited. The cities which have not yet been visited by ant k are indexed by $J_k(r)$. It is typical that the parameter β is negative so that shorter edges are favored. Linear dependence on $\tau(r,s)$ ensures preference is given to links that are well traversed (i.e. have a high pheromone level). The pheromone level on the selected edge is updated according to the local updating rule in Eq. (3).

$$\tau(r,s) \leftarrow (1-\rho) \cdot \tau(r,s) + \rho \cdot \tau_0 \tag{3}$$

Where:

ρ is the local pheromone decay parameter, $0 < \rho < 1$.
τ_0 is the initial amount of pheromone deposited on each of the edges.

Upon conclusion of an iteration (i.e. once all ants have constructed a tour), global updating of the pheromone takes place. Edges that compose the best solution are rewarded with an increase in their pheromone level. This is expressed in Eq. (4).

$$\tau(r,s) \leftarrow (1-\gamma) \cdot \tau(r,s) + \gamma \cdot \Delta\tau(r,s) \tag{4}$$

$$\Delta\tau(r,s) = \begin{cases} \frac{Q}{L} & \text{if } (r,s) \in \text{ globally best tour} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Where:

$\Delta\tau(r,s)$ is used to reinforce the pheromone on the edges of the global best solution (see Eq. (5)).
L is the length of the best (shortest) tour to date while Q is a constant.
γ is the global pheromone decay parameter, $0 < \gamma < 1$.

3. Accumulated Experience Ant Colony

The AEAC described herein is based on ACS, although it is possible to apply it to a range of ACO meta-heuristics. Essentially a new term w is incorporated into Eqs. (1) and (2) giving Eqs. (6) and (7) respectively.

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \left\{ w(r,u)\tau(r,u)[d(r,u)]^{\beta} \right\} & \text{if } q \leq q_0 \\ \text{Eq. (7)} & \text{otherwise.} \end{cases} \tag{6}$$

$$p_k(r,s) = \begin{cases} \frac{w(r,s)\tau(r,s)[d(r,s)]^\beta}{\sum_{u \in J_k(r)} w(r,u)\tau(r,u)[d(r,u)]^\beta} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

Given that $u \in J_k(r)$, $w(r,u)$ represents the relative weighting of link (r,u) compared to all other choices from node r, $0 \leq w(r,u) \leq 2$. This is calculated for all ants that have incorporated link (r,u) into their solution within the current iteration. If link (r,u) has been found to lead to longer paths after it has been incorporated into the solution, then the weighting $w(r,u) < 1$. On the other hand, if the reverse is the case, then $w(r,u) > 1$. If the colony as a whole has never incorporated link (r,u), its weighting is simply 1. Therefore, links that have historically been shown to induce shorter overall paths will be favored over those that have not. This represents a longer term approach than just using pheromone alone.

The value of w is updated at the beginning of each step for all links that are reachable by an ant during that step. This reduces computation time without affecting the algorithm's overall behavior, as it is not necessary to evaluate the probability (Eq. (7)) of links that cannot be considered by an ant. Before calculating w, the costs of the partial tours so far constructed are linearly scaled in the range [-1, 1], where -1 is the scaled value of the minimum length path and 1 is the corresponding value for the maximum length path. Then, for a given link (r,u), $w(r,u)$ is calculated by the following. The mean scaled cost of all paths that have used (r,u) is computed and, in the case of a minimization problem such as the TSP, subtracted from 1. Hence, if (r,u) has been used predominantly in shorter paths, the mean of the scaled costs will be closer to -1 and as such the value of $w(r,u)$ will be closer to 2. In the opposite situation, where a link has been used predominantly in longer paths, the correspondingly higher mean scaled cost will yield a value of w closer to 0. It is worth noting that as the number of ants that have used (r,u) approaches m, $w(r,u)$ will in general approach 1 given a uniform distribution of partial tour costs. This is because it becomes difficult to determine the individual contribution of (r,u) to the cost of solutions. The algorithm for updating w for all links is summarized in Fig. 1 while Eq. (8) is for a single link. The computational overhead associated with updating w is $O(n^2)$.

$$w(r,s) = \begin{cases} 1 - (|A(r,s)|)^{-1} \sum_{a \in A(r,s)} \frac{c(a) - c_{mid}}{c_{max} - c_{mid}} & \text{if } |A(r,s)| > 0 \\ 1 & \text{otherwise.} \end{cases} \quad (8)$$

Where:

c_{min} and c_{max} are the costs of the minimum and maximum partial tours respectively.
c_{mid} is the linear mid-point between c_{min} and c_{max}.
$c(a)$ is the cost of the partial tour constructed by ant a.
$A(r,s)$ is the set of all ants that have incorporated link (r,s) into their respective solutions.

```
Determine c_min and c_max
c_mid ← (c_max + c_min)/2
For each ant a
    c'(a) ← (c(a) − c_mid)/(c_max − c_mid)
End For
For each link (r, s) adjacent to an ant
    If |A(r, s)| > 0 Then
        c̄'_{a∈A(r,s)} ← mean of c'(a) for all a in A(r, s)
        w(r, s) ← 1 − c̄'_{a∈A(r,s)}
    Else
        w(r, s) ← 1
    End If
End For
```

Where $c'(a)$ is the scaled cost of the partial solution constructed by ant a.

Fig. 1. Algorithm to calculate w for each link.

4. An Improved Accumulated Experience Ant Colony

Initial testing of the AEAC showed that it performs as well or better than normal ACS on problems with less than 100 cities but significantly worse on problems with more than 200 cities. A number of possible reasons for this poor performance on large problems were considered. Underlying each of these reasons is the quadratic growth in links as the number of cities increases and hence, the large reduction in the number of links used within a single iteration. This reduces the proportion of links that can be assigned a meaningful value of w and thus may adversely bias the search away from some links and towards others through a number of mechanisms.

It was considered possible that the mean value of w for weighted links does not equal 1, which was confirmed by examination across a number of problems. However, a variation of AEAC in which unused links are assigned the mean value of w, rather than 1, was found to perform worse than the original. Another possibility is that although less information is available to calculate meaningful values of w on larger problems, the results of weighting these links takes immediate effect on ants' decisions with ensuing effects on pheromone levels due to local pheromone updates. Thus, the feedback from this could lead the colony to prematurely converge on a sub-optimal solution. Two techniques were considered to correct this problem. The first is to start weighting elements later in each iteration. This allows for a greater number of elements to be used before applying the knowledge gained to assign weights, while reducing feedback on the system from weighting a small number of elements from the beginning (when little is known about their relative utility). The second technique collects data about elements over a number of iterations. There are a number of ways this can be implemented, but the most effective technique we examined is to use old values of w until there is sufficient information to replace

Table 1. TSP instances used in this study.

Instance	Description	Optimal Cost
gr24	24 cities	1272
bays29	29 cities	2020
hk48	48 cities	11461
eil51	51 cities	426
berlin52	52 cities	7542
st70	70 cities	675
eil76	76 cities	538
pr76	76 cities	108159
kroA100	100 cities	21282
ch130	130 cities	6110
d198	198 cities	15780
kroA200	200 cities	29368
lin318	318 cities	42029
rd400	400 cities	15281
pcb442	442 cities	50778
att532	532 cities	27686

them. Thus, until an element has been used once, its value of w is 1. In subsequent iterations, if it has not been used, its value of w is updated according to Eq. (9).

$$w \leftarrow (1 - \rho_w) \cdot (w - 1) + 1 \tag{9}$$

Where:

> ρ_w is the rate at which the previous value of w ceases to influence future values.

This equation ensures that the influence of old values of w gradually decreases until it becomes close to 1 again. This is important because as time progresses, the colony will search in the neighborhood of different solutions and values of w derived from one neighborhood of solutions may not be as accurate for another neighborhood. When the element is used again, its value of w is recalculated according to Eq. (8). The combination of these two approaches produces the best improvement. For the experiments we use $\rho_w = 0.1$ and delay updates to w until 75% of the way through each iteration.

5. Computational Experience

A control strategy (normal ACS) and the two AEACs are run in order to evaluate their performance. Table 1 describes the TSP instances, from TSPLIB,[4] with which the AEAC meta-heuristic is tested.

The computing platform used to perform the experiments is a 550 MHz Linux machine. The computer programs are written in the C language. Each problem instance is run for 3000 iterations across 10 random seeds. The ACS parameter settings used are: $\beta = -2$, $\gamma = 0.1$, $\rho = 0.1$, $m = 10$, $q_0 = 0.9$.

5.1. *Results and Analysis*

Numerical results for the control and two AEACs are presented in Table 2. The minimum (**Min**), median (**Med**), maximum (**Max**) and inter-quartile range (**IQR**) are used to summarize the cost results, while only the median CPU time is shown given the highly consistent results for this measure. The relative percentage deviation (RPD) is used to express the results for cost. The RPD is a measure of the distance between a heuristic's solution cost and the optimal solution cost. It is calculated by Eq. (10). Fig. 2 shows graphically the median RPD cost by problem size.

$$\text{RPD} = 100 \times \left(\frac{c_{heur} - c_{opt}}{c_{opt}} \right) \tag{10}$$

Where:

c_{heur} is the cost of the heuristic solution.
c_{opt} is the optimal solution cost.

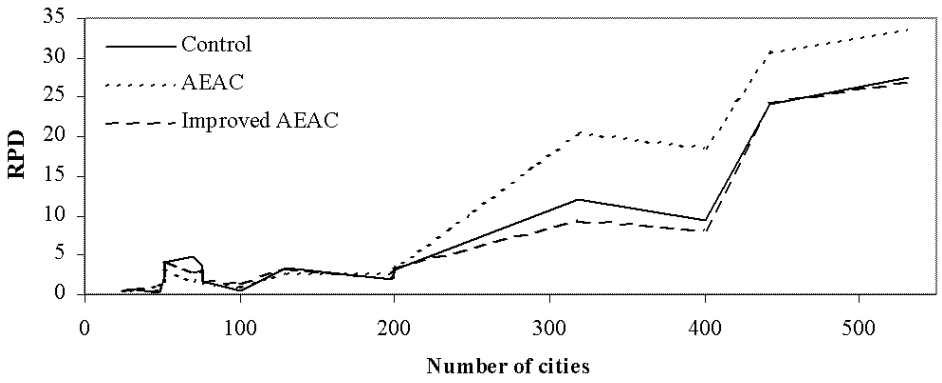

Fig. 2. Median RPD costs achieved by each strategy by problem size.

The original AEAC performed well on a number of problems, achieving lower minimum costs on `bays29`, `eil51`, `st70`, `pr76` and `ch130`, and equivalent minimum costs on `gr24`, `hk48` and `eil76` compared to the control strategy. It found the optimal solution on `gr24`, `bays29` and `hk48`. Above 100 cities, AEAC performed worse than the control strategy on all problems except `ch130`.

Given that the number of links grows with the square of the number of cities, it was considered likely that the original AEAC is unable to gather sufficient information about links to calculate meaningful values of w on larger problems. In large part this is because the original AEAC only accumulates experience over a single iteration. In an attempt to rectify this problem we created the improved AEAC.

Table 2. Results for control strategy and AEAC variants.

Strategy	Problem Instance	Cost (RPD) Min	Med	Max	IQR	CPU Time (seconds)
Control	gr24	0.0	0.5	4.4	0.1	18
	bays29	0.3	0.7	2.3	0.3	26
	hk48	0.0	0.3	3.4	0.7	71
	eil51	0.9	2.1	3.5	0.9	80
	berlin52	0.0	4.2	6.1	4.2	83
	st70	1.6	4.9	9.5	3.8	150
	eil76	1.5	3.6	4.8	0.9	198
	pr76	0.1	1.6	4.0	0.8	177
	kroA100	0.0	0.5	3.5	2.2	304
	ch130	2.5	3.5	7.0	2.1	520
	d198	1.1	2.0	3.0	1.3	1186
	kroA200	1.3	3.2	5.2	1.8	1231
	lin318	8.8	12.1	14.5	2.8	2972
	rd400	7.3	9.3	14.5	2.3	4946
	pcb442	20.1	24.1	27.7	3.2	6024
	att532	22.6	27.6	31.6	5.4	8758
AEAC	gr24	0.0	0.5	0.6	0.1	101
	bays29	0.0	0.5	0.9	0.3	111
	hk48	0.0	1.2	3.2	1.9	164
	eil51	0.2	1.4	2.6	1.3	175
	berlin52	0.1	3.3	8.0	2.8	178
	st70	0.4	1.8	5.5	2.4	258
	eil76	1.5	3.1	3.9	1.5	290
	pr76	0.1	1.5	4.2	1.5	289
	kroA100	0.4	1.1	3.0	0.5	442
	ch130	1.1	2.6	4.9	1.9	693
	d198	1.7	2.7	4.3	1.1	1493
	kroA200	1.4	2.8	6.0	2.0	1539
	lin318	11.1	20.5	26.7	5.0	3771
	rd400	17.3	18.6	31.4	1.2	5984
	pcb442	27.7	30.6	34.7	3.2	7272
	att532	31.0	33.6	37.4	2.4	10521
Improved AEAC	gr24	0.0	0.5	1.1	0.4	98
	bays29	0.6	0.7	2.3	0.2	106
	hk48	0.1	0.6	1.4	1.0	152
	eil51	0.5	2.2	5.4	1.9	162
	berlin52	2.5	4.3	6.9	1.8	165
	st70	0.4	3.0	5.0	1.5	237
	eil76	0.2	2.9	4.1	1.9	265
	pr76	1.3	1.7	3.2	0.7	263
	kroA100	0.0	1.4	3.9	2.3	401
	ch130	1.5	3.3	4.7	1.0	629
	d198	1.6	2.1	2.6	0.7	1357
	kroA200	0.8	3.3	6.1	1.6	1403
	lin318	7.5	9.5	12.3	3.2	3473
	rd400	6.8	8.3	11.2	1.2	5612
	pcb442	22.0	24.3	29.1	2.4	6801
	att532	24.6	27.1	28.5	2.1	9921

The improved AEAC performs significantly better than the original. It also produces better solutions than the control on a number of problem instances, including many of the larger problems. It achieved lower minimum costs on `kroA200`, `lin318`, and `rd400` and a better median cost on `att532`. On problems with less than 100 cities it is not as successful as the original AEAC, but in these problems the original AEAC does not suffer from a lack of information concerning many links and the use of old values of w may be more harmful than useful.

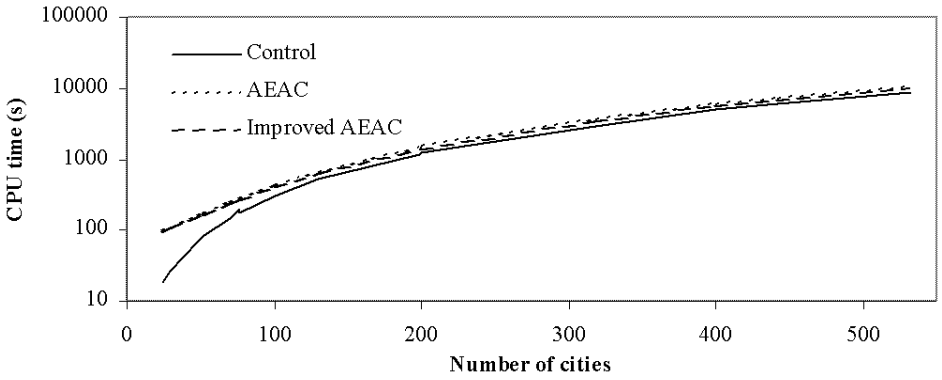

Fig. 3. Median CPU time used by each strategy by problem size.

The computational overhead associated with calculating w adversely affects the CPU time used by AEAC. On all problems both AEACs had longer CPU times than the control strategy. Fig. 3 shows the median CPU time graphically. It uses a logarithmic scale for CPU time to provide more information about the time difference on smaller problems, where the difference is most evident. Since the procedure for determining w only considers the reachable links at each step, the difference in CPU time between the control and AEACs decreases with the inverse square of the number of cities. Further experiments were carried out in which the control strategy was run for the same amount of time as the improved AEAC, but its results were still inferior.

6. Conclusions

ACS is a highly greedy search algorithm that can benefit from techniques that consider the long-term effects of decisions made at each step and hence, make it less greedy. The AEAC approach relies on historical information and patterns developed over time in order to calculate which element to add to a solution at a particular step of the algorithm. The original AEAC performed most successfully on the smaller problems (those with less than 100 cities), finding the optimal solution to three and equaling or improving upon the best solution found by the control

strategy on most others. As the number of cities exceeds 100 AEAC's performance becomes poorer, probably because as the number of cities increases, the number of links used within each iteration decreases quadratically. Because AEAC relies on information gathered as links are used, this problem can greatly impact on its performance on larger problems.

An improved AEAC was also created in which information from previous iterations is used when no information is available during the current iteration. This multiple-iteration form of the algorithm produces improved results, especially on larger problems. Compared to a normal ACS implementation, the improved AEAC is able to perform equally well on most problems, and better on a number of the largest problems. The improved AEAC does not perform as well as the original AEAC on problems with less than 100 cities, suggesting that on these problems accumulating experience over a single iteration is sufficient to yield improved results.

In the future, we wish to investigate the possibility of achieving the same effects by less computationally intensive means using the normal pheromone updating rules. A similar but complementary approach is described by Montgomery and Randall.[5] The AEAC is part of a wider strategy that is looking at ways of producing generic strategies to enhance ant based meta-heuristics.[6,7] Our next step is to determine how well AEAC applies across a range of problems, particularly those in which there is no obvious relation between adjacent elements.

References

1. M. Heusse, S. Guérin, D. Snyders and P. Kuntz, Adaptive agent-driven routing and load balancing in communications networks, *Advances in Complex Systems* **1** (1998) 237-254.
2. M. Dorigo and G. Di Caro, The ant colony optimization meta-heuristic, in *New Ideas in Optimization*, eds. D. Corne, M. Dorigo and F. Glover (McGraw-Hill, London, 1999) 11–32.
3. M. Dorigo and L. Gambardella, Ant colony system: A cooperative learning approach to the traveling salesman problem, *IEEE Trans. Evol. Comput.* **1** (1997) 53–66.
4. G. Reinelt, TSPLIB - A traveling salesman problem library, *ORSA J. Comput.* **3** (1991) 376–384.
5. J. Montgomery and M. Randall, Anti-pheromone as a Tool for Better Exploration of Search Space, *Proc. 3rd Int. Workshop on Ant Algorithms, ANTS2002*, Brussels, Belgium, September 2002.
6. J. Montgomery and M. Randall, Alternative pheromone applications for ant colony optimisation, Technical Report TR02-07, School of Information Technology, Bond University, Australia.
7. M. Randall, A general framework for constructive meta-heuristics, *Operations Research/Management Science at Work*, (Kluwer Academic Publishers, Boston, 2002) 111–128.