

# **cAS: Ant Colony Optimization with Cunning Ants**

Shigeyoshi Tsutsui

Hannan University, Matsubara Osaka 580-8502, Japan,  
tsutsui@hannan-u.ac.jp

**Abstract.** In this paper, we propose a variant of an ACO algorithm called the *cunning* Ant System (*cAS*). In *cAS*, each ant generates a solution by borrowing a part of a solution which was generated in previous iterations, instead of generating the solution entirely from pheromone density. Thus we named it, cunning ant. This *cunning* action reduces premature stagnation and exhibits good performance in the search. The experimental results showed *cAS* worked very well on the TSP and it may be one of the most promising ACO algorithms.

## **1 Introduction**

As a bio-inspired computational paradigm, ant colony optimization (ACO) has been applied with great success to a large number of hard problems. They include the traveling salesman problem (TSP) [1–3], the quadratic assignment problem [4], scheduling problem [5], and vehicle routing problem [6], among others.

The first ACO algorithm was called the Ant System (AS) [1], and is applied to the TSP. Since then, many advanced ACO algorithms are proposed as extensions of AS. Typical of these are AS with elitist strategy and ranking (AS<sub>rank</sub>) [6], Ant Colony System (ACS) [2], and MAX-MIN Ant System (MMAS) [3]. These advanced ACO algorithms include a strong exploitation of the best solutions found during the search. However, strong exploitation causes premature stagnation of the search. The most successful ones, such as ACS and MMAS, have explicit features to avoid this premature stagnation [7]. Thus for developing a successful variant of ACO, we need to incorporate a good balance between exploitation and exploration.

In this paper, we propose a variant of an ACO algorithm called the *cunning Ant System* (*cAS*). In *cAS*, each ant generates a solution by borrowing a part of a solution from a previous iteration, instead of generating the solution entirely from pheromone density. From this behavior, we call them *cunning ants*. This *cunning* action reduces premature stagnation and exhibits good performance in the search.

Although the basic background is different, the idea of using partial solutions to seed the ants' solution construction is inspired by our previous study on the edge histogram based sampling algorithm (EHBSA) [8] within the EDA [9] framework for permutation domains. Using partial solutions to seed solution

construction in ACO framework has been performed by combining an external memory implementation in [10, 11]. In [12], some solution components generated according to ACO are removed, resulting in a partial candidate solution. Starting from the partial solution, a complete candidate solution is reconstructed by a greedy construction heuristic.

In the remainder of this paper, Section 2 gives a brief overview of MMAS. Then, Section 3 describes how the solutions with *cAS* are constructed, and the empirical analysis is given in Section 4. Finally, Section 5 concludes this paper.

## 2 A Brief Review of MMAS

Since *cAS* uses the MMAS framework in pheromone density updating, in this section we give a brief overview of MMAS.

MMAS allows the deposit of pheromone by either the *iteration-best*, or *best-so-far* ant to introduce a strong exploitation feature in the search. To counteract the stagnation caused by this, MMAS introduced an important mechanism to limit the possible range of pheromone trail density within the interval  $[\tau_{min}, \tau_{max}]$ . By limiting the influence of the pheromone trails we can avoid the relative differences between the pheromone trails from becoming too extreme during the run of the algorithm. MMAS also introduced schemes of pheromone trail reinitialization and/or pheromone trail smoothing (PTS) to prevent stagnation of the search. In MMAS, the values of  $\tau_{max}$  and  $\tau_{min}$  are defined as

$$\tau_{max}(t) = 1/(1 - \rho) \times 1/C_t^{best-so-far}, \quad (1)$$

$$\tau_{min}(t) = \frac{\tau_{max} \cdot (1 - \sqrt[n]{p_{best}})}{(n/2 - 1) \cdot \sqrt[n]{p_{best}}}, \quad (2)$$

where  $C_t^{best-so-far}$  is the fitness of *best-so-far* solution at  $t$  and  $n$  is the problem size and  $p_{best}$  is a control parameter. With a smaller value of  $p_{best}$ , the value of  $\tau_{min}$  becomes larger.

## 3 Cunning Ant System (*cAS*)

### 3.1 Cunning ant

In traditional ACO algorithms, each ant generates a solution probabilistically or pseudo-probabilistically based on the current pheromone trail  $\tau_{ij}(t)$ . In this paper, we introduce an agent called *cunning ant* (*c-ant*). The *c-ant* differs from traditional ants in the manner of solution construction. It constructs a solution by borrowing a part of existing solutions. The remainder of the solution is constructed based on  $\tau_{ij}(t)$  probabilistically as usual. In a sense, since this agent in part appropriates the work of others to construct a solution, we named the agent *c-ant* after the metaphor of its cunning behavior. In the remainder of this paper a solution constructed by a *c-ant* is also represented with the same notation, *c-ant*. Also, an agent which has constructed a solution borrowed by a *c-ant* is called a *donor ant* (*d-ant*) and the solution is also represented with the notation *d-ant*.

Fig. 1 shows an example of the relationship between *c-ant* and *d-ant* in TSP. Here note again the notations *c-ant* and *d-ant* are used both for agents and solutions. In this example, the *c-ant* borrows part of the tour,  $7 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 3$ , from the *d-ant* directly. The *c-ant* constructs the remainder of the tour for cities 4, 5, and 6 according to  $\tau_{ij}(t)$  probabilistically. Using *c-ant* in this way, we can prevent premature stagnation the of search, because only a part of the cities in a tour are newly generated, and this can prevent over exploitation caused by strong positive feedback to  $\tau_{ij}(t)$  (see Section 4.2).

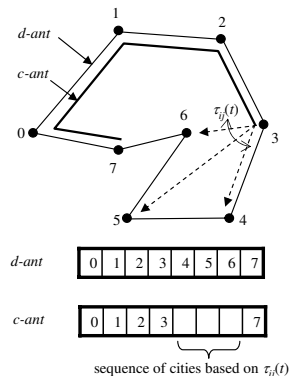


Fig. 1. *c-ant* and *d-ant* in TSP

### 3.2 Colony model of cAS

In *cAS*, we use a colony model as shown in Fig. 2, which is similar to the colony model proposed for real parameter optimization with ACO framework [13]. It consists of  $m$  units. Each unit consists of only one  $ant_{k,t}^*$  ( $k = 1, 2, \dots, m$ ). At iteration  $t$  in unit  $k$ , a new  $c-ant_{k,t+1}$  creates a solution with the existing ant in the unit (i.e.,  $ant_{k,t}^*$ ) as the  $d-ant_{k,t}$ . Then, the newly generated  $c-ant_{k,t+1}$  and  $d-ant_{k,t}$  are compared, and the better one becomes the next  $ant_{k,t+1}^*$  of the unit.

Thus, in this colony model,  $ant_{k,t}^*$ , the best individual of unit  $k$ , is always reserved. Phomone density is then updated with  $ant_{k,t}^*$  ( $k=1, 2, \dots, m$ ) and  $\tau_{ij}(t+1)$  is obtained as:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \sum_{k=1}^m \Delta^* \tau_{ij}^k(t), \quad (3)$$

$$\Delta^* \tau_{ij}^k(t) = 1/C_{k,t}^* : \text{if } (i, j) \in ant_{k,t}^*, 0 : \text{otherwise}, \quad (4)$$

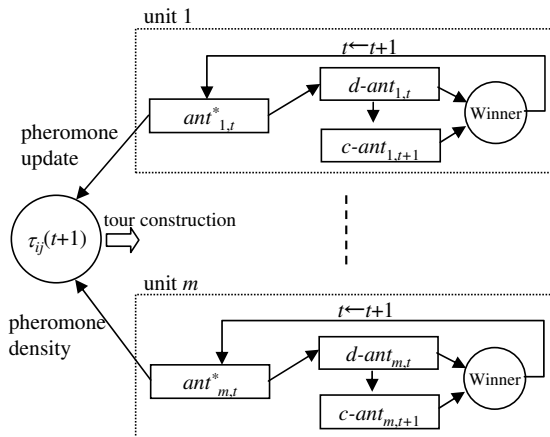


Fig. 2. Colony model of *cAS*

where  $C_{k,t}^*$  is the fitness of  $ant_{t,k}^*$ .

In  $cAS$ , pheromone update is performed with  $m$   $ant_{k,t}^*$  ( $k=1,2,\dots,m$ ) by Eq. 3 within  $[\tau_{min}, \tau_{max}]$  as in MMAS [3]. Here,  $\tau_{max}$  for  $cAS$  is defined as

$$\tau_{max}(t) = \frac{1}{1-\rho} \times \sum_{k=1}^m \frac{1}{C_{k,t}^*}, \quad (5)$$

and  $\tau_{min}$  is given by Eq. 2 of MMAS. Here, note that  $\tau_{max}$  of Eq. 5 is obtained by modifying Eq. 1 of MMAS.

In this colony model, a  $d-ant$  is the best ant of each unit. By using  $ant^*$  as a  $d-ant$  in each unit we can expect an appropriate level of exploitation. Further, the comparison method in each sub-colony is similar to the tournament selection in genetic algorithms (GAs), being well known that tournament selection can maintain diversity of a population, though it differs from traditional tournament selection in that the comparison restricted to being performed inside of each unit. Thus, we can also expect this colony model to maintain the diversity of  $ant_k^*$  in the system.

### 3.3 Number of sampling and borrowing nodes

A crucial question when  $c-ant$  creates a new solution is how to determine which part of the solution the  $c-ant$  will borrow from the  $d-ant$ . To ensure robustness across a wide spectrum of problems, it should be advantageous to introduce variation both in the portion and the number of nodes of the partial solution that is borrowed from  $d-ant$ . First it is reasonable to choose the starting node position of the partial solution randomly. Thereafter, the number of nodes of the partial solution must be determined. Let us represent the number of nodes that are constructed based on  $\tau_{ij}(t)$ , by  $l_s$ . Then,  $l_c$ , the number of nodes of partial solution, which  $c-ant$  borrows from  $d-ant$ , is  $l_c = n - l_s$ . Here, let us introduce a control parameter  $\gamma$  which can define  $E(l_s)$  (the average of  $l_s$ ) by  $E(l_s) = n \times \gamma$ .

In previous studies [8], when generating a permutation string, part of the permutation elements were copied from a *template* string. To determine the sampling portion in a string, we used the  $c$  cut-point approach. We sampled nodes for only one randomly chosen segment from  $c$  segments obtained by applying  $c$  cut points to the template. With this approach,  $l_s$  distributes in the range  $[0, n]$ , and  $E(l_s) = n \times 1/c$ . Thus, with  $c$  cut-point method above,  $E(l_s)$  is  $n/2, n/3, \dots$  for  $c = 2, 3$ , and so on, and,  $\gamma$  corresponds to  $1/c$ , i.e.,  $\gamma$  can take only the values of 0.5, 0.333, and 0.25, corresponding to  $c = 2, 3, 4$  and so on.

In the current research, we extend this elementary method to a more flexible technique which allows for  $\gamma$  taking values in the range  $[0.0, 1.0]$ . The probability density function of  $l_s$  with the  $c$  cut-point approach is [14]:

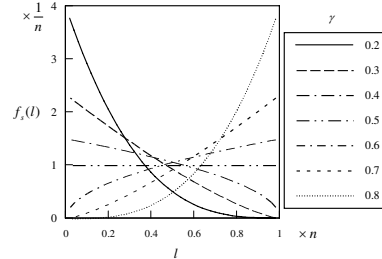
$$f_s(l) = \frac{(c-1)}{n} \left(1 - \frac{l}{n}\right)^{c-2}, \quad 0 < l < n, \quad c \geq 2. \quad (6)$$

Here, we extend  $c$  so that it can take a continuous value ( $c \geq 2$ ). Then, we can obtain a generalized  $f_s(l)$  by setting  $c = 1/\gamma$  in Eq. 6 for  $0.5 \geq \gamma > 0$  as follows:

$$f_s(l) = \frac{1-\gamma}{n\gamma} \left(1 - \frac{l}{n}\right)^{\frac{1-2\gamma}{\gamma}}. \quad (7)$$

For  $0.5 < \gamma < 1$ , we further extend the above logic as follows. First we consider distribution of  $l' = n-l$  and  $p'_s = 1-\gamma$  in Eq. 7. Then we can obtain the following equation for  $0.5 < \gamma < 1$ .

$$f_s(l) = \frac{\gamma}{n(1-\gamma)} \left(\frac{l}{n}\right)^{\frac{2\gamma-1}{1-\gamma}}. \quad (8)$$



**Fig. 3.** Distribution of  $l_s$  for various  $\gamma$

Fig. 3 shows  $f_s(l)$  for  $\gamma = 0.2, 0.3, 0.4, 0.5, 0.6, 0.7$ , and  $0.8$ . We can see from this figure that for a smaller  $\gamma$ , shorter lengths of  $l_s$  become dominant, and for a larger  $\gamma$ , longer lengths of  $l_s$  become dominant.

The algorithm of *cAS* is summarized in Fig. 4.

1.  $t \leftarrow 0$
2. Set the initial density  $\tau_{ij}(t) = C$  (an arbitrary large value, e.g. 10)
3. Sample two individuals randomly for each unit  $k$ , then choose the best one in the unit and set it as  $ant_{k,0}^*$  ( $k=1,2,\dots,m$ )
4. Update  $\tau_{ij}(t)$  according to Eq. 3 with  $\tau_{max}, \tau_{min}$  of Eqs. 5 and 2
5. Sample  $c\text{-}ant_{k,t+1}$  for  $k=1,2,\dots,m$  according to  $d\text{-}ant_{k,t}$  ( $: ant_{k,t}^*$ ) and  $\tau_{ij}(t+1)$
6. Compare  $c\text{-}ant_{k,t+1}$  and  $d\text{-}ant_{k,t}$ , set the best one as  $ant_{k,t+1}^*$  for  $i=1,2,\dots,m$
7.  $t \leftarrow t+1$
8. If the termination criteria are met, terminate the algorithm. Otherwise, go to 4

**Fig. 4.** Algorithm description of *cAS*

## 4 Experiments

Here we evaluate *cAS* on TSP. Unless explicitly indicated otherwise, the following default parameter settings are used, which are the same values as used with MMAS in [3] except for  $p_{best}$ , i.e.,  $\alpha = 1, \beta = 2$  ( $\alpha$  and  $\beta$  are parameters that control the relative importance of the trail and the heuristic value [3]),  $m = n$  ( $m$  is the number of units and  $n$  is the number of cities), a candidate list [3] with a size of 20. For *cAS*,  $p_{best}$  value of 0.005 and  $\gamma$  value of 0.4 were used. All test instances are taken from TSPLIB.

### 4.1 Performance of *cAS*

The performance of *cAS* was compared with MMAS and ACS, both of which outperform other existing ACO algorithms [7]. The comparison was performed on the same number of tour constructions for all algorithms as is described in [15]

and [3]; this number was chosen as  $k \times n \times 10000$ , where  $k = 1$  for symmetric TSPs and  $k = 2$  for asymmetric TSPs (ATSPs). 25 runs were performed. Performance of each algorithm was compared using  $Best_{avg}$  (average of the best tour length) and  $Error$  (average excess rate from optimum length) over 25 runs.

Table 1 summarizes the results. The results of MMAS+PTS and MMAS are taken from [3] and those of ACS are from [15]. We also showed the results of *non-cAS*; i.e., we use colony model shown in Fig. 2 but no cunning action is applied. This correspond to *cAS* with  $\gamma = 1$ . The values in bold show the best performance for each instance. From this table, we can see that *cAS* outperforms almost all instances used in the experiments except for d198. Further, we can observe that even *non-cAS* has similar performance to MMAS and thus the effectiveness of using the colony model in Fig 2 is also confirmed.

**Table 1.** Results of *cAS*  $Best_{avg}$  is average best solution over 25 runs and  $Error$  indicates average excess (%) of  $Best_{avg}$  from optimal in 25 runs.

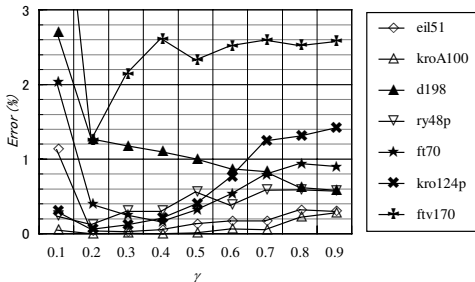
TSP	opt	<i>cAS</i>						MMAS				ACS		
		<i>cAS</i> ( $\gamma=0.4$ )			<i>non-cAS</i> ( $\gamma=1$ )			MMAS+pts		MMAS		ACS		
		$Best_{avg}$	$std^*$	$Error$ (%)	$Best_{avg}$	$std^*$	$Error$ (%)	$Best_{avg}$	$Error$ (%)	$Best_{avg}$	$Error$ (%)	$Best_{avg}$	$Error$ (%)	
STSP	eil51	426	<b>426.2</b>	0.5	<b>0.06</b>	427.3	0.7	0.31	427.1	0.26	427.6	0.38	428.1	0.48
	kroA100	21282	<b>21282.0</b>	0.0	<b>0.00</b>	21332.4	48.5	0.24	21291.6	0.05	21320.3	0.18	21420.0	0.65
	d198	15780	<b>15954.1</b>	35.6	<b>1.10</b>	15958.5	10.9	1.13	15956.8	1.12	15972.5	1.22	16054.0	1.74
ATSP	ry48p	14422	<b>14465.4</b>	34.9	<b>0.30</b>	14509.5	46.7	0.61	14523.4	0.70	14553.2	0.91	14565.5	0.99
	ft70	38673	<b>38736.1</b>	77.1	<b>0.16</b>	39105.8	169.5	1.12	38922.7	0.65	39040.2	0.95	39099.1	1.10
	kro124p	36230	<b>36303.2</b>	120.3	<b>0.20</b>	36734.1	261.1	1.39	36573.6	0.95	36773.5	1.50	36857.0	1.73
	ftv170	2755	2827.1	8.7	2.62	2820.6	14.8	2.38	<b>2817.7</b>	<b>2.28</b>	2828.8	2.68	2826.5	2.59

\*  $std$ : standard deviation of  $Best_{avg}$

## 4.2 Parameter values for $\gamma$

Fig. 5 shows the variations of  $Error$  for various  $\gamma$  values. Here,  $\gamma$  values were varied starting from 0.1 to 0.9 with step 0.1. Except for d198 and ftv170,  $\gamma$  values of [0.2, 0.6], which are in the smaller value range of  $\gamma$ , showed good performance. On ftv170, *cAS* with  $\gamma$  values of 0.2 showed good performance. On d198,  $\gamma$  values of [0.4, 0.9], which are in the larger value range of  $\gamma$ , showed good performance.

Fig. 6 shows the convergence process of change of  $Error$  on kroA100 (100-city symmetric TSP) for  $\gamma$  values of 0.1, 0.3, 0.5, 0.7, and 0.9. Early stagnations of search can be observed with  $\gamma$  values of 0.7 and 0.9. With  $\gamma$  values of 0.3 and 0.5, stagnations of search occur much later in the search. With a  $\gamma$  value of 0.1, no stagnation can be observed. But the convergence process is very slow.



**Fig. 5.** Variation of  $Error$  for  $\gamma$

Thus we can see that using appropriate small values of  $\gamma$  can prevent over exploitation with strong positive feedback to  $\tau_{ij}(t)$  and lead to success searches.

### 4.3 Parameter values for $\rho$

With the ACO scheme, parameter  $\rho$  also plays an important role in controlling the search process. With a larger value of  $\rho$ , the search proceeds slowly, but it prevents the stagnation of the search. On the other hand, with a smaller value of  $\rho$ , the search proceeds rapidly, but it causes stagnation.

In *cAS*, as seen in Section 4.2, parameter  $\gamma$  has an effect similar to that of  $\rho$ . Fig. 7 shows the effects of

variations of the value of  $\rho$  on the quality of solutions on kroA100 with  $\gamma$  values of 0.2 (left) and 0.6 (right). With  $\gamma = 0.6$ , though the performance is bad compared with  $\gamma = 0.2$ ,  $\rho$  having a strong effect. On the other hand, with  $\gamma = 0.2$ , we can see the effect of variation of  $\rho$  is weaker compared with  $\gamma = 0.6$ . But still an appropriate value of  $\rho$  ( $\rho=0.98$ ) leads to successful runs. Thus, we can see the synergy effect of parameters  $\gamma$  and  $\rho$ .

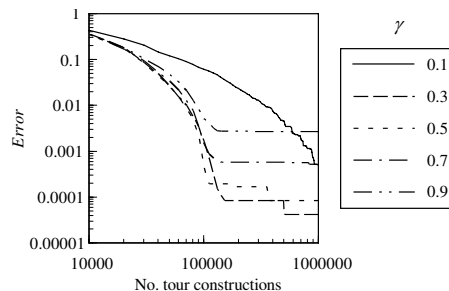


Fig. 6. Convergence process on kroA100

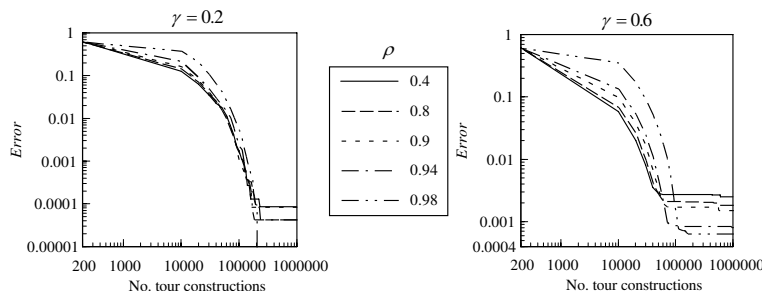


Fig. 7. Effect of  $\rho$  values on kroA100. *Error* is average over 25 runs.

### 4.4 Improving performance of *cAS* with local search

Here we study *cAS* with a local search on symmetrical TSP. One of the best performing local searches for TSP is the well-known Lin-Kernighan algorithm (LK) [16]. The implementation of LK is complex compared with 2-OPT and 3-OPT heuristics. There are many variant implementations for LK. An important, and widely adopted scheme is the repeated use of the basic LK algorithm. The scheme is referred to as Chained Lin-Kernighan [17], or Iterated Lin-Kernighan [18]. In addition to the basic LK, Chained LK repeatedly utilizes a method for perturbing a given tour which is called *kick*. We used a Chained LK called

*Concorde TSP solver* (Concorde) developed by D. Applegate et.al, which is available for research purposes at [19]. *cAS* is written in JAVA and Concorde is written in C. So we combined it with *cAS* using Java Native Interface (JNI). Concorde was compiled using *MinGW* on Windows XP. For each tour generated by *cAS*, we applied it  $n$  iterations of Chained LK with *random-walk kicks*, which is reported to have the best performing *kick* [17].

The following six instances, which range in hundreds and thousands of cities, were used: att532, d1291, vm1748, pr2392, fl3795, and rl5934. The maximum execution time ( $T_{max}$ ) of the *cAS* with LK for each instance is set to 40, 80, 200, 240, 1400, and 3300 seconds, respectively. The machine we used had two Opteron 275 (2.4GHz) processors, 2GB main memory, and 32-bit WindowsXP. For unit size,  $m = 5$  was used for all instances. For other parameters, we used  $\rho = 0.5$ .  $\gamma = 0.4$ . For  $\tau_{min}$ , we used  $\tau_{min} = \tau_{min}/2n$  to attain somewhat tighter bounds on the allowed pheromone trail strength according to the recommendation in [3].

To confirm the effectiveness of combining *cAS* with LK, we also tested the following three algorithms: *non-cAS* with LK (i.e.,  $\gamma=1$ , see Section 4.1), MMAS with LK, and Chained LK alone. For MMAS, we used our implementation with Java.  $\rho$  value of 0.8 was used. For MMAS, we tuned by testing all combinations of  $m = \{5, 10\}$  and pheromone update strategy = {iteration-best, best-so-far, and the schedule described in [3] for use with LK}. The results of the combination of  $\{m = 5\} \times \{\text{pheromone update strategy} = \text{best-so-far}\}$  scored the best  $\#OPT$  on bigger problems (i.e., pr2392, fl3795, and rl5934) and we used this combination for MMAS with LK. We ran Concorde iterating the basic LK with *random-walk kicks* until the achieving time defined by  $T_{max}$ . In the Chained LK, the initial tour affects the performance. In this experiment, we chose the *Quick-Borka* tour which has good performance on medium runs [17].

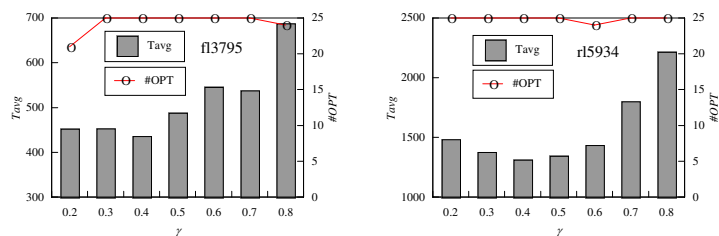
Table 2 summarizes the results. We can see all algorithms of *cAS*, *non-cAS*, and MMAS showed very small values of *Error* by combining LK and thus the advantage of combining these algorithms with LK is very clear. However, when we focus our attention on the results of  $\#OPT$ , all algorithms except for *cAS* could not attain  $\#OPT = 25$  for d1291,vm1748, pr2392, fl3795, and rl5934. In contrast to this, *cAS* could attain  $\#OPT = 25$  for all test instances within the allowed run time  $T_{max}$  showing the smallest  $T_{avg}$  (average time in seconds to find optimal in successful runs) among algorithms tested. Here we notice again that even *non-cAS* shows very similar  $\#OPT$  results to MMAS, as were observed without local search in Section 4.1. Thus, we can see that the effectiveness of using the combination of the proposed colony model and *c-ant* holds true for *cAS* with local search also. Fig. 8 shows the variations of *Error* for various  $\gamma$  values. Here,  $\gamma$  values were varied starting from 0.2 to 0.8 with step 0.1. We can see that *cAS* with  $\gamma$  values within range [0.3, 0.5] shows small  $T_{avg}$  with  $\#OPT = 25$ .

## 5 Conclusions

In this paper, we proposed the *cAS*, a new ACO algorithm, and evaluated the performance using TSP instances available at TSPLIB. The results showed that

**Table 2.** Results of *cAS* with LK on symmetrical TSP.  $I_{avg}$  and  $T_{avg}$  are average iterations and time in second to find optimal in successful runs. *Error* indicates average excess (%) from optimal in 25 runs.

TSP	<i>cAS</i>												$T_{max}$					
	<i>cAS</i> ( $\gamma=0.4$ )						<i>non-cAS</i> ( $\gamma=1$ )							MMAS			Chained LK	
	LdO#	Error (%)	$I_{avg}$	$T_{avg}$ [min, max]	LdO#	Error (%)	$I_{avg}$	$T_{avg}$ [min, max]	LdO#	Error (%)	$I_{avg}$	$T_{avg}$ [min, max]		LdO#	Error (%)	$T_{avg}$ [min, max]		
att532 ( $n=532$ )	25	0.00	1.8	7.8 [1.4, 27.8]	24	0.00	1.9	8.2 [1.4, 32.9]	25	0.00	2.4	10.5 [1.4, 32.6]	17	0.02	6.11 [0.3, 28.5]	40		
d1291 ( $n=1291$ )	25	0.00	5.7	27.4 [6.0, 54.4]	24	0.00	7.4	35.9 [6.0, 56.9]	22	0.00	10.3	48.8 [6.1, 74.1]	6	0.12	17.0 [4.0, 61.3]	80		
vm1748 ( $n=1748$ )	25	0.00	5.6	72.4 [8.4, 171.0]	24	0.00	5.6	77.5 [8.1, 169.8]	21	0.00	5.6	78.4 [8.3, 173.0]	1	0.06	72.8 [-]	200		
pr2392 ( $n=2392$ )	25	0.00	10.1	104.9 [33.7, 190.0]	24	0.00	13.4	137.2 [57.3, 205.9]	12	0.00	20.4	211.3 [170.3, 233.2]	4	0.17	122.4 [40.2, 222.1]	240		
fl3795 ( $n=3795$ )	25	0.00	9.8	435.1 [102.8, 1228.7]	15	0.00	13.9	615.9 [119.4, 1138.2]	17	0.00	17.6	770.7 [159.9, 1081.1]	0	0.57	-	1400		
rl5934 ( $n=5934$ )	25	0.00	43.2	1336.1 [729.1, 1996.8]	1	0.00	59.6	1854.6 [-]	10	0.00	82.8	2533.6 [1499.2, 2897.0]	0	0.27	-	3300		



**Fig. 8.** Variations of  $T_{avg}$  and  $\#OPT$  for various values on fl3795 and rl5934.

*cAS* worked well on the test instances and has performance that may be one of the most promising ACO algorithms. We also evaluate *cAS* when it is combined with LK local search heuristics using larger sized TSP instances. The results also showed promising performance.

*cAS* introduced two important schemes. One is to use the colony model divided into units, which has a stronger exploitation feature while maintaining a certain degree of diversity among units. The other is to use a scheme, we call cunning, when constructing new solutions, which can prevent premature stagnation by reducing strong positive feedback to the trail density.

However, we need more analytical study on the relationships between these schemes and traditional schemes with ACO, including the further tuning of competitor algorithms. To apply *cAS* to other applications, such as the scheduling problem and the quadratic assignment problems, also remains for future work.

## Acknowledgements

The author would like to acknowledge to the reviewers for their insightful comments on improving this paper. This research is partially supported by the Min-

istry of Education, Culture, Sports, Science and Technology of Japan under Grant-in-Aid for Scientific Research number 16500143.

## References

1. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on SMC-Part B* **26**(1) (1996) 29–41
2. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE TEC* **1**(1) (1997) 53–66
3. Stützle, T., Hoos, H.: Max-min ant system. *Future Generation Computer Systems* **16**(9) (2000) 889–914
4. Gambardella, L.M., Taillard, E.D., Dorigo, M.: Ant colony for the quadratic assignment problem. *J. of the Operational Research Society* **50** (1999) 167–176
5. Stützle, T.: An ant approach to the flowshop problem. *Proc. of the 6th European Congress in Intelligent Thech. and Com.* **3** (1998) 1560–1564
6. Bullnheimer, B., Hartl, R.F., Strauss, C.: An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research* **89** (1999) 319–328
7. Dorigo, M., Stützle, T.: *Ant Colony Optimization*. MIT press, Massachusetts (2004)
8. Tsutsui, S.: Probabilistic model-building genetic algorithms in permutation representation domain using edge histogram. *Proc. of the 7th Int. Conf. on Parallel Problem Solving from Nature (PPSN VII)* (2002) 224–233
9. Pelikan, M., Goldberg, D.E., F.Lobo: A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications* **21**(1) (2002) 5–20
10. Acan, A.: An external memory implementation in ant colony optimization. *Proc. of the Fourth International Workshop on Ant Algorithms and Swarm Intelligence (ANTS-2004)* (2004) 73–84
11. Acan, A.: An external partial permutations memory for ant colony optimization. *Proc. of the 5th European Conference on Evolutionary Computation in Combinatorial Optimization* (2005) 1–11
12. Wiesemann, W., Stützle, T.: An experimental investigation of iterated ants for the quadratic assignment problem. *IRIDIA Technical Report Series Technical Report TR/IRIDIA/2006-003* (2006)
13. Tsutsui, S.: An enhanced aggregation pheromone system for real-parameter optimization in the aco metaphor. *Proc. of the Fifth International Workshop on Ant Algorithms and Swarm Intelligence (ANTS-2006)* (2006)
14. Tsutsui, S., Pelikan, M., Goldberg, D.E.: Using edge histogram models to solve permutation problems with probabilistic model-building genetic algorithms. *IlliGAL Report No. 2003022* (2003)
15. Gambardella, L.M., Dorigo, M.: Solving symmetric and asymmetric tsps by ant colonies. *Proc. the IEEE Int. Conf. on Evo. Comp.(ICEC'96)* (1996) 622–627
16. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling salesman problem. *Operations Research* **21** (1973) 498–516
17. Applegate, D., Cook, W., Rohe, A.: Chained lin-kernighan for large traveling salesman problems. *INFORMS J. on Computing* **15** (2003) 82–92
18. Johnson, D.S.: Experimental analysis of heuristics for the stsp. G. Gutin and A. P. Punnen (ed.), *The Traveling Salesman Problem and Variations* **9** (2002) 369–443
19. <http://www.tsp.gatech.edu/concorde.html>. Concorde tsp solver, Ansi c code as gzipped tar file (2005)