Missouri Estimation of Distribution Algorithms Laboratory

cAS: The Cunning Ant System

Shigeyoshi Tsutsui and Martin Pelikan

MEDAL Report No. 2007007

May 2007

Abstract

This paper describes the cunning ant system (cAS), which can be used to optimize problems with candidate solutions represented by permutations. cAS generates new candidate solutions from a combination of the pheromone density and the archive of past candidate solutions. Performance of cAS is demonstrated on two important classes of permutation problems: (1) the traveling salesman problem and (2) the quadratic assignment problem. In both problem classes, cAS is shown to perform very well and outperform other techniques for solving these problems. The paper also discusses how the efficiency of cAS can be further improved by combining cAS with local search and by parallelizing cAS using a master-slave architecture. Finally, the paper presents a pheromone-density entropy measure, which can be used to measure the potential of cAS and other ACO algorithms to generate novel, fitter solutions.

Keywords

Ant colony optimization, cunning ant system, traveling salesman problem, quadratic assignment problem, hybridization, parallelization.

Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Department of Mathematics and Computer Science
University of Missouri–St. Louis
One University Blvd., St. Louis, MO 63121
E-mail: medal@cs.umsl.edu
WWW: http://medal.cs.umsl.edu/

*c*AS: The Cunning Ant System

Solving the traveling salesman problem and the quadratic assignment problem using the cunning ant system

Shigeyoshi Tsutsui[1], **Martin Pelikan**[2]

[1] Department of Management and Information Science
Hannan University
Matsubara, Osaka 580-8502, Japan
tsutsui@hannan-u.ac.jp

[2] Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Department of Mathematics and Computer Science
University of Missouri–St. Louis, 320 CCB
One University Blvd.
St. Louis, MO 63121, U.S.A.
pelikan@cs.umsl.edu

Abstract This paper describes the cunning ant system (*c*AS), which can be used to optimize problems with candidate solutions represented by permutations. *c*AS generates new candidate solutions from a combination of the pheromone density and the archive of past candidate solutions. Performance of *c*AS is demonstrated on two important classes of permutation problems: (1) the traveling salesman problem and (2) the quadratic assignment problem. In both problem classes, *c*AS is shown to perform very well and outperform other techniques for solving these problems. The paper also discusses how the efficiency of *c*AS can be further improved by combining *c*AS with local search and by parallelizing *c*AS using a master-slave architecture. Finally, the paper presents a pheromone-density entropy measure, which can be used to measure the potential of *c*AS and other ACO algorithms to generate novel, fitter solutions.

1 Introduction

As a bioinspired computational paradigm, ant colony optimization (ACO) has been applied with great success to a broad range of optimization problems, including the traveling salesman problem [1–3], vehicle routing [4],

scheduling [5], the quadratic assignment problem [6,7,3], and optimization of continuous problems [8,9].

In this paper we describe the cunning ant system (cAS), which can be applied to optimization problems defined over permutations. To ensure a proper balance between exploration and exploitation and avoid premature stagnation, cAS generates new candidate solutions using a combination of the pheromone density and the archive of past candidate solutions. Specifically, a part of each new candidate solution is acquired from one of the solutions visited in the past and the remaining components are generated from the pheromone density, which is continuously updated based on the solutions visited during the run.

We demonstrate the performance of cAS on two important classes of permutation-based problems: (1) the traveling salesman problem (TSP) and (2) the quadratic assignment problem (QAP). To improve cAS performance, we combine cAS with local search techniques, such as the robust tabu search (for QAP), and the Lin-Kernighan algorithm (for TSP). Combining cAS with local search proves to be highly beneficial and to significantly improve cAS performance. For both TSP and QAP, cAS is shown to outperform other ACO approaches as well as other popular approaches for solving these problems. Nonetheless, the application of cAS is not limited to QAP and TSP; cAS can be applied to any problem where candidate solutions are represented by permutations.

The paper starts with a brief overview of ACO for permutation-based problems in Section 2. Next, Section 3 describes the basic procedure of cAS with the focus on solving the traveling salesman problem (TSP). Section 4 discusses the application of cAS to standard TSP benchmarks and compares the performance of cAS to that of other techniques for solving TSP. Section 5 shows how cAS can be adapted to the quadratic assignment problem (QAP) and analyzes the performance of cAS on standard QAP benchmarks. Section 7 discusses efficiency enhancement of cAS and proposes a simple master-slave approach to parallelization of cAS. Finally, Section 8 summarizes and concludes the paper.

2 Background

Research on algorithms inspired by the behavior of ants started with the ant system (AS) [1,10], which was applied to the traveling salesman problem (TSP), a well-known NP-complete optimization problem. The task in TSP is to find the shortest path in a given graph that passes through all nodes of this graph.

AS guides its search for the shortest path by simulating a system of agents (ants), which walk through the set of cities of a given TSP problem instance. Each direct path (i, j) between two connected cities i and j is assigned a pheromone density $\tau_{i,j}$, which represents the desirability of this path. Initially, all paths are assigned the same pheromone density.

Ants walk through the set of cities using a probabilistic rule that ensures that paths with a higher pheromone density are preferred to those with a lower density. Each ant deposits pheromones on its path. The amount of pheromone deposited depends on the overall length of the path; shorter paths are given more pheromones than the longer ones. As a result, paths that are contained in short tours are expected to contain higher pheromone density than others.

Since shorter paths are given more pheromones than longer paths and ants prefer paths with higher pheromone densities, AS is expected to generate shorter and shorter tours over time, until the shortest tour is discovered. The pheromones deposited by ants evaporate over time, so the paths that are used infrequently are expected to get less attractive over time.

The ant colony system (ACS) [2] improves AS to ensure a better balance between exploration and exploitation. ACS uses two types of pheromone updates: (1) the global update rule, which is applicable only to the best tour found so far, and (2) the local update rule, which is used to update the pheromone density of each generated tour. In addition to modifying the pheromone updates, ACS uses local search to improve its performance. ACS was shown to provide significantly better performance than AS on TSP [2].

In the max-min ant system (MMAS) [3], the pheromones are deposited by either the *iteration-best* or the *best-so-far* ant to introduce a strong exploitation feature in the search for shortest paths. To counteract the stagnation that may be caused by the strong selection pressure, MMAS introduced an important mechanism to limit the possible range of pheromone trail densities to remain within the interval $[\tau_{min}, \tau_{max}]$. By limiting the range of pheromone densities we can prevent the relative differences between the pheromone densities from becoming too extreme. 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 τ_{max} and τ_{min} are defined as

$$\tau_{\max}(t) = \frac{1}{1-\rho} \cdot \frac{1}{f_{best-so-far}}, \tag{1}$$

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

where $f_{best-so-far}$ is the cost of the best solution found so far, n is the problem size, and p_{best} is a control parameter. With a smaller value of p_{best}, the value of τ_{min} becomes larger.

*c*AS uses the MMAS framework [3] to update its pheromone densities and to limit the range of their values (with small modifications) as is described in the next section.

3 Cunning Ant System (*c*AS)

In traditional ACO algorithms, each ant generates a solution probabilistically based on the current pheromone densities $\tau_{ij}(t)$. *c*AS uses agents called

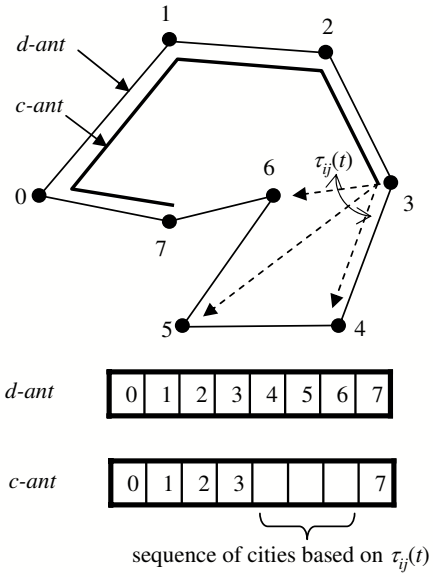

Fig. 1 An example of a cunning ant borrowing a part of its solution from a previously generated tour in TSP. In TSP, the cunning ant always samples a contiguous block of positions. In this example, the cunning ant borrows part of the tour, 7→0→1→2→3, from the donor directly. The cunning ant then constructs the remainder of the tour for cities 4, 5, and 6 according to the pheromone densities probabilistically.

cunning ants, which differ from traditional ants in the manner of solution construction. A part of each new solution is taken from one of the previously generated solutions (also called a donor) whereas the remainder of the solution is generated probabilistically from the pheromone densities. In a sense, since this agent in part appropriates the work of others to construct a solution, we named the agent a *cunning* ant after the metaphor of its cunning behavior. Using partial solutions in solution construction in ACO can be found also in [11–13].

In the first part of this paper, we restrict the sampling to one contiguous block of positions (remaining positions are acquired from the donor). Sampling contiguous blocks of positions should be beneficial especially for problems that contain strong nearest-neighbor dependencies, such as the traveling salesman problem. For an example, see Figure 1. Later, Sect. 5 discusses how the sampling can be modified for problems with strong dependencies between the positions and their values but with weak dependencies between the nearest neighbors, which is the case, for example, in the quadratic assignment problem.

3.1 Colony model of cAS

In *c*AS, we use an elitist colony model, which is similar to the colony model proposed for real parameter optimization with ACO framework [8,14]. In this model we maintain an *archive* consisting of m candidate solutions (permutations) generated in the past; kth solution in the archive at iteration t is denoted by $s_{k,t}$ ($k \in \{1, 2, \ldots, m\}$).

At iteration t, a new solution is generated for each position k in the archive using the current solution in this position, $s_{k,t}$ as the donor. Then, the newly generated candidate solution is compared with its donor with respect to the objective function (e.g. tour length for TSP or solution cost for QAP), and the best of the two survives as the next solution in this position of the archive, $s_{k,t+1}$.

Using elitism in maintaining the archive provides an important source of selection pressure and ensures that *c*AS preserves the best solutions found so far. Since the solutions in different positions of the archive do not compete with each other, *c*AS can also be expected to preserve diversity and avoid sampling solutions from only one region of the search space. A combination of these two mechanisms provides a robust mechanism for finding a proper balance between exploration and exploitation as is also supported by empirical results.

3.2 Combining donor solutions with sampling from pheromones

A crucial question when a cunning ant creates a new solution is how to determine which part of the solution the ant will borrow from its donor. To ensure robustness across a wide spectrum of problems, it should be advantageous to introduce variation both in the number of nodes to borrow from the donor as well as in the way these nodes are selected. As mentioned earlier, we start by assuming that the sampling will be restricted to one contiguous block of positions, which should be beneficial for TSP and other problems in which nearest-neighbor dependencies are strong. Sect. 5 discusses another sampling approach, where any positions are allowed to be sampled.

In our previous work concerned with TSP [15], we determined the positions to sample using the so-called c cut-point approach, which was inspired by k-point crossover. In this approach, c positions (cut points) were first selected randomly and then one of the segments between two consequent cut points was selected randomly for sampling. The remaining positions were selected from a template.

With the c cut-point approach, the length of a segment, l_s is distributed in the range $[0, n]$, and the expected length of a sampled segment is $E[l_s] = n/c$. The probability density function of the segment length using the c cut-point approach is given by [15]

$$pdf(l_s) = \frac{(c-1)}{n}\left(1 - \frac{l}{n}\right)^{c-2}, \; 0 < l < n, \; c \geq 2. \tag{3}$$

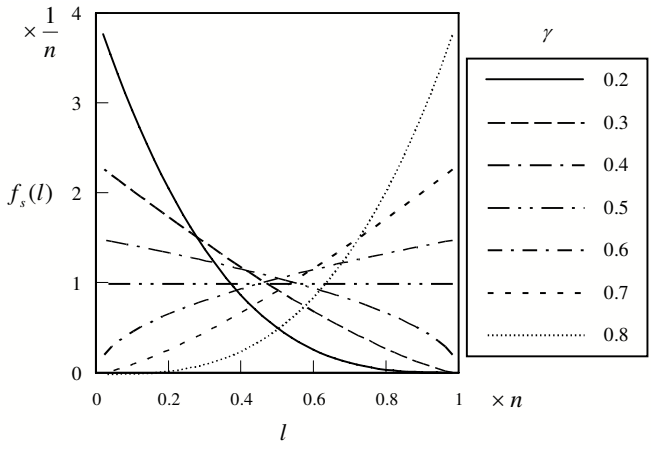

Fig. 2 Distribution of l_s for various values of γ. The greater the parameter γ, the greater the expected number of sampled positions.

In cAS, we use a somewhat more flexible approach. Just like in the previous work, the sampling from pheromone densities is done for one contiguous block of nodes (permutation elements). The first position of the block of nodes to sample is chosen randomly with the uniform distribution over all positions. The length of the block (number of nodes) is sampled from a probability distribution to ensure that the average number of nodes to sample is $\gamma \times n$ where $\gamma \in (0, 1]$ is a control parameter given by the user, which specifies the average proportion of nodes to sample. The probability distribution used to sample the segment length is inspired by the cut-point approach used in our previous work, but it allows for an arbitrary value of γ:

$$f_s(l) = \begin{cases} \frac{1-\gamma}{n\gamma}\left(1-\frac{l}{n}\right)^{\frac{1-2\gamma}{\gamma}} & \text{for } \gamma \in (0, 0.5]. \\ \frac{\gamma}{n(1-\gamma)}\left(\frac{l}{n}\right)^{\frac{2\gamma-1}{1-\gamma}} & \text{for } \gamma \in (0.5, 1]. \end{cases} \tag{4}$$

For $\gamma = 1/c$, the new approach is identical to the c cut-point approach. However, the new approach allows the user to set γ arbitrarily, unlike in the c cut-point approach.

Fig. 2 shows the distribution of segment lengths for different values of γ. The figure shows that for a smaller γ, shorter lengths of l_s become dominant, and for a larger γ, longer lengths of l_s become dominant.

3.3 Generating new permutations from pheromone densities

To generate new candidate solutions in TSP, we use the standard probabilistic rule used in other ACO algorithms for TSP [1,3]. Just like the procedures for selecting the subset of positions to sample, also the rules

used to generate new permutations (tours) differ for TSP and QAP slightly due to the different definitions of these problems. Here we describe the rule used for TSP; Sect. 5 discusses the rule used for QAP.

To generate a new tour for TSP, the ant must first decide on the starting node. Here due to the use of cunning ant agents, the starting node is given by the node that precedes the first position we plan to sample. Let's now assume that the previous node we sampled is node i. The probability of sampling the node j as the next node is given by [1,3]

$$p_{i,j} = \frac{\tau_{i,j}^{\alpha} d_{i,j}^{-\beta}}{\sum_{k \in N(i)} \tau i, j^{\alpha} d_{i,j}^{-\beta}} \tag{5}$$

where $\alpha, \beta > 0$ are control parameters, $N(i)$ denotes the set of possible successors of node i (all nodes that have not yet been included in the generated solution) and $d_{i,j}$ denotes the length of the path from node i to node j.

3.4 Pheromone updates

Once a new candidate solution has been generated for each position k in the archive and the archive has been updated, the candidate solutions from the updated archive are used to update the pheromone densities. For all pairs of nodes i and j, the new pheromone density $\tau_{ij}(t+1)$ on the path between these nodes is obtained as

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t+1), \tag{6}$$

where

$$\Delta\tau_{ij}^{k}(t+1) = \begin{cases} \frac{1}{f(s_{k,t+1})} & \text{if } (i,j) \in s_{k,t+1} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where $f(s_{k,t+1})$ is the value of the objective function (tour length or solution cost) of $s_{k,t+1}$, and $\rho \in [0,1)$ is the pheromone evaporation rate. Note that here we consider *minimization* of the objective function as is usual in TSP.

The pheromone update is performed with all m candidate solutions in the archive according to Eq. 6, keeping all pheromone densities within the interval $[\tau_{min}, \tau_{max}]$ as in MMAS [3]. The upper bound τ_{max} for *c*AS is defined using a modification of the original formula for the upper bound from [3] (see Eq. 1) as

$$\tau_{\max}(t) = \frac{1}{1-\rho} \times \sum_{k=1}^{m} \frac{1}{f(s_{k,t})}, \tag{8}$$

The lower bound τ_{min} is defined the same as in MMAS [3] (see Eq. 2).

The *c*AS algorithm is summarized in Fig. 3. Note that *c*AS can be applied to any problem for which candidate solutions are represented by permutations. Examples of such problems are the quadratic assignment problem and the traveling salesman problem, which are both discussed in this paper.

1. $t \leftarrow 0$
2. Set the initial pheromone density $\tau_{ij}(t) = C$ (an arbitrary large value, e.g. 10)
3. Sample two solutions (tours) randomly for each position $k \in \{1, 2, \ldots, m\}$ of the archive, the best of the two solutions in pos. k is stored in the archive as $s_{k,0}$.
4. Update $\tau_{ij}(t)$ according to Eq. 6 with τ_{max}, τ_{min} defined in equations 8 and 2
5. Sample a new tour $s'_{k,t}$ for all $k = 1, 2, \ldots, m$ using $s_{k,t}$ as the donor.
6. Compare $s_{k,t}$ with $s'_{k,t}$, set $s_{k,t+1}$ to be the best of the two for all $k = 1, 2, \ldots, m$.
7. $t \leftarrow t+1$
8. If the termination criteria are met, terminate the algorithm. Otherwise, go to 4.

Fig. 3 Pseudocode of the cunning ant system (cAS).

The differences between the different problem classes are threefold: (1) the method for selecting the positions to sample (for TSP we sample contiguous blocks of positions unlike for QAP and most other problems), (2) the probabilistic rule for generating new permutations (for TSP the rule exploits pairwise distances between the cities unlike for QAP and other problems), and (3) the pheromone update rule used to update the pheromone densities (in TSP the pheromone densities are assigned to pairs of neighbors unlike in QAP and most other problems). The choice of the best approach to use depends on the problem type. If the problem contains strong dependencies between nearest neighbors in the permutation, it is preferable to use cAS designed for TSP. On the other hand, if the absolute-position dependencies are strong, it is preferable to use cAS designed for QAP.

4 Traveling Salesman Problem (TSP)

4.1 Problem definition

A problem instance of the traveling salesman problem (TSP) [16] is given by a graph of n nodes with edges that correspond to paths between the nodes. Each edge is assigned a distance (cost). The task is to find a shortest (cheapest) path through all nodes, which starts and terminates in the same node.

There are two standard types of TSP instances: (1) symmetric, where for each pair of nodes the path cost between these two nodes is the same in both directions, and (2) asymmetric, where the path cost between any two nodes is not guaranteed to be the same in both directions. TSP is NP-complete and it is thus widely believed that TSP cannot be solved in polynomial time.

4.2 Tested instances

To test *c*AS on TSP, we used a number of TSP benchmarks from TSPLIB [17], considering problems of 48 to 5934 cities. Specifically, we considered the following instances for the compared algorithms without any local search: ry48p, eil51, ft70, kroA100, kro124p, ftv170, and d198. For the algorithms with LK local search, we considered the following, larger instances: att532, d1291, vm1748, pr2392, fl3795, and rl5934.

4.3 cAS on TSP without local search

Most parameters used here are the same as those used in MMAS [3] except for p_{best}. More specifically, $\alpha = 1$, $\beta = 2$, and $m = n$ (m is the size of the archive and n is the number of cities). The size of the candidate list is 20 [3]. That means that for any city the next city to visit is chosen from the 20 closest cities using the pheromone densities; only if all 20 closest cities have already been visited, other cities are considered and the next city is selected as the one that maximizes $\tau_{ij}^{\alpha} d_{ij}^{-\beta}$. Finally, $p_{best} = 0.005$ and $\gamma = 0.4$ (other values of γ are tested in subsection 4.4).

The performance of *c*AS was compared with MMAS and ACS, both of which outperform other existing ACO algorithms [18]. For all algorithms the overall number of tour constructions was upper bounded by $k \times n \times 10000$ ($k = 1$ for symmetric TSP, $k = 2$ for asymmetric TSP) as in [19,3]. For each problem instance, 25 independent runs were performed. Performance of all algorithms was compared using the average of the best tour length denoted by $Best_{avg}$ and the average excess rate from the global optimum denoted by $Error$. The results for MMAS+PTS and MMAS are taken from [3], whereas the results for ACS are taken from [19]. We also show the results for *c*AS without the use of cunning agents (donors) where all new solutions are created only by sampling, which corresponds to $\gamma = 1$ (this algorithm is denoted by *non-c*AS).

The results are summarized in Table 1. The values in bold show the best performance for each instance. From this table, we can see that *c*AS outperforms other MMAS and ACS in all instances except for ftv170. Furthermore, we can observe that even *non-c*AS has similar performance as MMAS, which provides further support for the effectiveness of using the colony model of *c*AS. Nonetheless, using cunning agents proves to be strongly beneficial.

4.4 Effect of γ on cAS performance on TSP

Recall that γ specifies the proportion of nodes to sample from pheromone densities (the remaining nodes are taken from the donor). To investigate the effects of γ on performance of *c*AS, we examined the performance of *c*AS for different values of $\gamma \in [0.1, 0.9]$ (see Fig. 4). Except for d198 and ftv170, γ values in the range of $[0.2, 0.6]$ lead to best performance. On ftv170, best

Table 1 Performance of *c*AS, MMAS and ACO on TSP. $Best_{avg}$ is the average tour length of the best solutions obtained in 25 independent runs and *Error* indicates the average excess (in %) of $Best_{avg}$ from the global optimum.

	TSP	opt	*c* AS						MMAS				ACS	
			c AS (γ=0.4)			*non-c* AS (γ=1)			MMAS+pts		MMAS			
			$Best_{avg}$	*std**	*Error* (%)	$Best_{avg}$	*std**	*Error* (%)	$Best_{avg}$	*Error* (%)	$Best_{avg}$	*Error* (%)	$Best_{avg}$	*Error* (%)
STSP	eil51	426	**426.2**	0.5	**0.06**	427.3	0.7	0.31	427.1	0.26	427.6	0.38	428.1	0.48
	kroA100	21282	**21282.0**	0.0	**0.00**	21332.4	48.5	0.24	21291.6	0.05	21320.3	0.18	21420.0	0.65
	d198	15780	**15954.1**	35.6	**1.10**	15958.5	10.9	1.13	15956.8	1.12	15972.5	1.22	16054.0	1.74
ATSP	ry48p	14422	**14465.4**	34.9	**0.30**	14509.5	46.7	0.61	14523.4	0.70	14553.2	0.91	14565.5	0.99
	ft70	38673	**38736.1**	77.1	**0.16**	39105.8	169.5	1.12	38922.7	0.65	39040.2	0.95	39099.1	1.10
	kro124p	36230	**36303.2**	120.3	**0.20**	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	**2817.7**	**2.28**	2828.8	2.68	2826.5	2.59

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

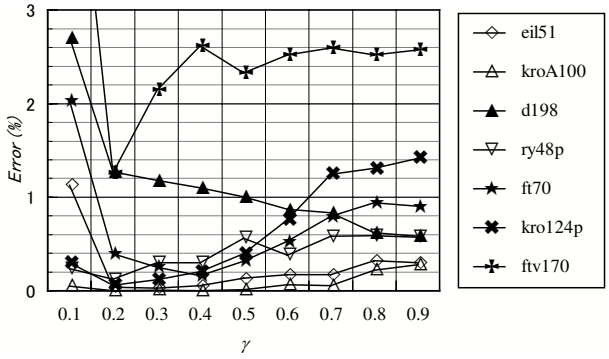

Fig. 4 Effect of γ on the quality of solutions obtained with *c*AS on TSP.

performance was achieved with $\gamma = 0.2$. On d198, larger values of γ from the range of $[0.4, 0.9]$ perform best.

4.5 Measuring evolvability using pheromone entropy on TSP

One of the important issues when studying properties of any stochastic optimization technique is whether the technique is still capable of creating novel solutions or the technique has practically converged and it is unlikely that it will reach better solutions than those generated thus far. To describe this property, evolutionary biologists and other researchers often use the term *evolvability* [20]. One of the definitions of evolvability is that a system is called evolvable if its properties show heritable genetic variation, and if natural selection can thus change these properties [20]. Since *c*AS with some settings showed signs of premature stagnation, we decided to define *pheromone entropy* to measure evolvability of *c*AS and to be able to identify when no more improvement of the best-so-far solution can be expected.

More specifically, we define the entropy $I(t)$ of the pheromone density $\tau_{ij}(t)$ at time t as follows:

$$I(t) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j\neq i} p_{ij}(t), \tag{9}$$

where $p_{ij}(t)$ is the normalized pheromone density at time t, defined as

$$p_{ij}(t) = \frac{\tau_{ij}(t)}{\sum_{j\neq j}\tau ij(t)}. \tag{10}$$

The upper bound of $I(t)$ is obtained when all pheromone densities are equal, which is true in the first iteration of cAS and is very unlikely to happen later in the run. For TSP, the upper bound on $I(t)$ is defined as [21]

$$High(I(t)) = I(0) = \log(n-1). \tag{11}$$

As the pheromone densities concentrate around specific promising tours, the entropy $I(t)$ is expected to decrease. The entropy $I(t)$ reaches its lower bound when the likelihood of generating a single solution (or two solutions in the case of symmetric TSP) is maximized and the likelihood of all other solutions is minimized [21,22]. This situation happens when the pheromone densities of one single solution (or two solutions in the case of symmetric TSP) are all τ_{max} and the remaining pheromone densities are τ_{min}. The lower bound for the symmetric TSP is given by [21]

$$Low_S(I(t)) = \log(2r+n-3) - \frac{2r\log r}{2r+n-3}. \tag{12}$$

and for the asymmetric TSP it is [21]

$$Low_A(I(t)) = \log(r+n-2) - \frac{r\log r}{r+n-1}, \tag{13}$$

where $r = \tau_{max}/\tau_{min}$.

Given the upper and lower bounds on the entropy, we can compute the normalized entropy $I_N(t)$, which takes only values from $[0,1]$ [21]:

$$I_N(t) = \frac{I(t) - Low(I(t))}{High(I(t)) - Low(I(t))}. \tag{14}$$

To better understand the influence of γ on the balance between exploration and exploitation as well as the meaning of the normalized pheromone entropy, Fig. 5 visualizes both the *Error* as well as the normalized entropy $I_N(t)$ with respect to the time on the symmetric TSP instance eil51 for $\gamma \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. For larger values of γ, early stagnation can be observed (for the number of tour constructions of around 90,000 to 200,000), which is also supported by the corresponding normalized entropy, which reaches its lower bound when the runs start stagnating. On the other hand, for $\gamma = 0.1$, the best solution steadily improves even later in the run, as is also supported by the normalized entropy, which remains relatively large even late in the run. An analogical result for an asymmetric TSP instance kro124p is shown in Fig. 6.

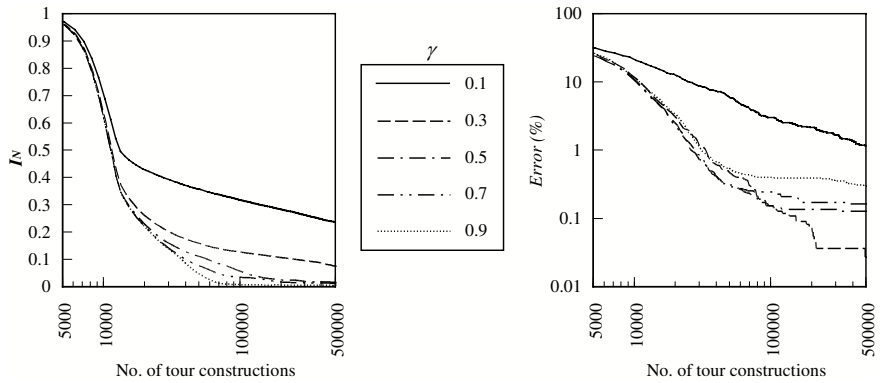

Fig. 5 Early stagnation and the normalized pheromone entropy for cAS on eil51 (symmetric TSP).

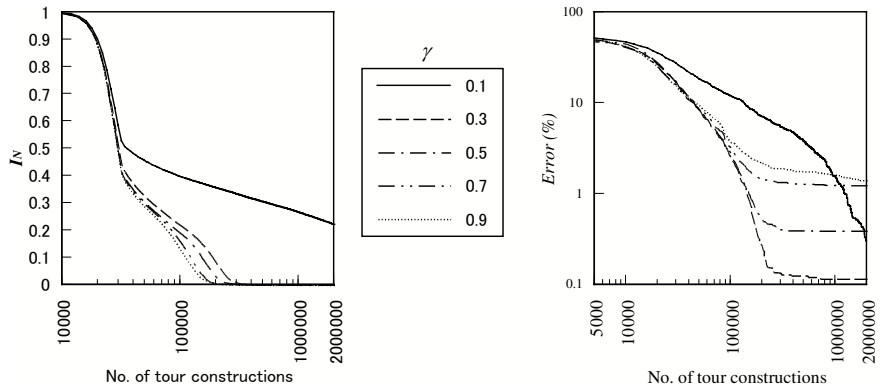

Fig. 6 Early stagnation and the normalized pheromone entropy for cAS on kro124p (asymmetric TSP).

4.6 Effect of ρ on performance of cAS on TSP

In ACO, the parameter ρ (pheromone evaporation rate) plays an important role in controlling the search process. With a larger value of ρ, the search proceeds slowly, but it is less likely to get stuck in a local optimum. On the other hand, with a smaller value of ρ, the search proceeds more rapidly, but it is also more likely to get stuck in a local optimum. Note that in cAS, γ has an effect similar to that of ρ (just like ρ, γ also influences the balance between exploration and exploitation).

Fig. 7 shows the effects ρ on the quality of solutions obtained with cAS on kroA100 for $\gamma = 0.2$ and $\gamma = 0.6$. The figure shows that the effects of ρ on cAS performance become stronger with larger γ. With smaller γ, exploration is already relatively slow and the effects of ρ are not as substantial.

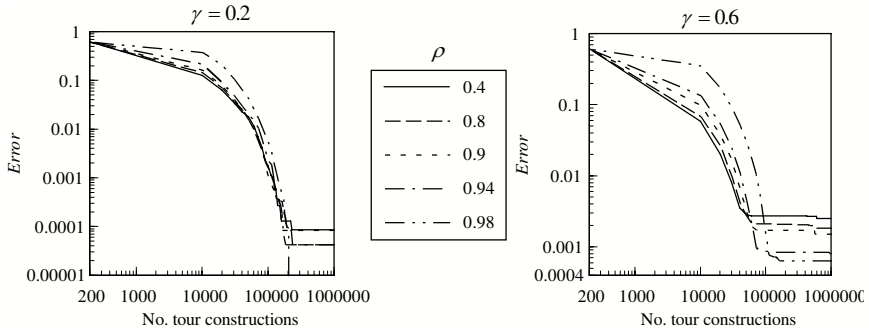

Fig. 7 Effect of ρ on the errors obtained on kroA100.

4.7 Improving performance of cAS on TSP with local search

In real-world applications of evolutionary algorithms, practitioners must often use various efficiency enhancement techniques [23] to further improve performance of these algorithms. One of the most popular approaches to enhancing the efficiency of evolutionary algorithms is hybridization where a local search technique is incorporated into the evolutionary algorithm to locally improve candidate solutions. In this section we show that incorporating local search into *c*AS significantly improves performance of this algorithm and allows larger problems to be solved in practical time.

One of the best performing local search techniques for TSP is the well-known Lin-Kernighan algorithm (LK) [24]. The implementation of LK is relatively complex compared to the simpler 2-OPT or 3-OPT heuristics and many variants of LK have been proposed in the past. Chained LK [25] and iterated LK [26] are among the most successful LK implementations. We used the chained LK implementation called *Concorde* developed by D. Applegate et al. [27]. Since our implementation of *c*AS was written in Java and Concorde was implemented in ANSI C, we used Java Native Interface to bridge the two codes. The LK algorithm was applied to each tour generated by *c*AS and in each application of LK, the algorithm was allowed to perform n iterations. AS the perturbation method in LK, we used *random-walk kicks*, which was reported to yield best performance of LK [25]. Using chained LK with random-walk kicks in *c*AS allowed us to solve much larger instances than without any local search.

To provide a fair comparison between the different techniques, we limited the overall execution time that the algorithms can use to solve each instance. The maximum execution time (T_{max}) is set to 40 (for att532), 80 (for dl281), 200 (for vm1748), 240 (for pr2392), 1400 (for fl3795), and 3300 (for rl5934) seconds, respectively. All experiments were run on a computer with two Opteron 275 (2.4GHz) processors, 2GB main memory, and 32-bit Windows XP. For all instances, the archive of $m = 5$ solutions was used. The remaining parameters were set as follows: $\rho = 0.5$ and $\gamma = 0.4$.

Based on the recommendation from [3], we used $\tau_{min}/2n$ as the lower bound for pheromone densities to attain somewhat tighter bounds on the allowed pheromone trail strength.

To confirm the effectiveness of combining cAS with LK, we also tested the following three algorithms: *non-c*AS with LK (i.e., $\gamma = 1$), MMAS with LK, and chained LK alone. All experiments used $\rho = 0.8$. For MMAS, we empirically tuned the algorithm by trying two settings for m (5 and 10), and three pheromone update strategies from [3] (iteration-best, best-so-far, and the schedule proposed in [3] for the use with the LK heuristic). The best combination of parameters was $m = 5$ with the best-so-far pheromone update strategy. As the initialization strategy for LK, we chose the *Quick-Borka* tour.

The results of this comparison are summarized in Table 2. We can see that the LK algorithm improved performance of all algorithms in terms of solution quality. Nonetheless, for the most difficult problem instances, all algorithms except for cAS failed to find the global optimum in 100% of cases and their performance appeared to be more significantly affected by the increasing problem size. Furthermore, cAS obtained the solutions in the shortest average running time. Thus, cAS was both more reliable as well as more efficient. The results also confirm that *non-c*AS performs similarly as MMAS as was already shown for the tests without local search.

Even with LK, we analyzed the influence of γ on cAS performance; the results of these experiments are summarized in Fig. 8. The results indicate that cAS with $\gamma \in [0.3, 0.5]$ shows a relatively small average time to convergence while retaining reliable convergence in all 25 independent runs.

Table 2 Performance of cAS with LK, MMAS with LK, and ACO with LK on symmetrical TSP. I_{avg} is the average number of iterations until the global optimum and T_{avg} indicates the average time (in seconds) until the global optimum (considering only successful runs). *Error* indicates the average excess (in %) from the global optimum over the 25 runs.

TSP	*c* AS								MMAS				Chained LK			T_{max}
	c AS (γ=0.4)				*non-c* AS (γ=1)											
	#OPT	*Error* (%)	I_{avg}	T_{avg} [min, max]	#OPT	*Error* (%)	I_{avg}	T_{avg} [min, max]	#OPT	*Error* (%)	I_{avg}	T_{avg} [min, max]	#OPT	*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

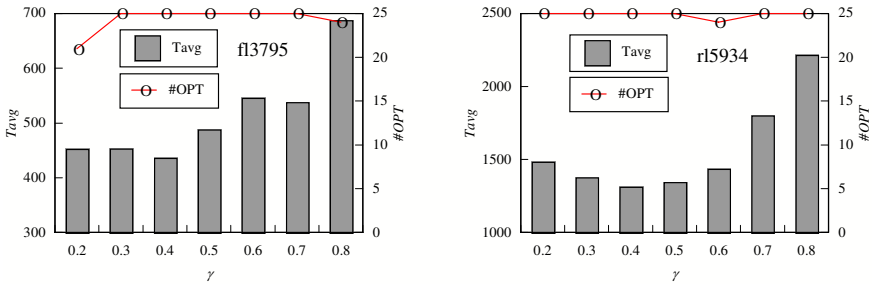

Fig. 8 Effect of γ on the average time (T_{avg}) and the number of successful runs (#OPT) on TSP instances fl3795 and rl5934.

5 Quadratic Assignment Problem (QAP)

5.1 Problem description

In the quadratic assignment problem (QAP) [28], we are given n locations and n facilities and the task is to assign the facilities to the locations to minimize the cost. For each pair of locations i and j, we are given their distance $d_{i,j}$. For each pair of facilities i and j, we are given the flow $f_{i,j}$ between these facilities (regardless of their locations). The cost of each assignment of facilities to locations is given by the sum over all pairs of locations of the product of the distance between these locations and the flow between the facilities assigned to these locations.

Assignments of facilities to locations can be encoded as permutations; the facility at location i is given by the value at position i of the permutation. The cost for an assignment (permutation) ϕ that is to be minimized is given by

$$cost(\phi) = \sum_{i \neq j} d_{i,j} f_{\phi(i),\phi(j)}. \tag{15}$$

QAP is also an NP-complete problem. Several ACO approaches to solving QAP were proposed in the past [6,7,3].

5.2 Adapting cAS to QAP

While in both TSP and QAP the candidate solutions are represented by permutations, the structure of these problems is quite different. More specifically, in TSP, solution quality is influenced only by the pairs of cities that directly follow each other; on the other hand, in QAP, it does not really matter what facilities follow each other but solution quality is influenced by absolute positions of the different facilities, the flows between these positions (locations), and the corresponding distances. To incorporate this knowledge, we modified the original *c*AS slightly as is described next.

First of all, to select positions to sample, we no longer restrict the sampled positions to form one contiguous block of positions but distribute the positions randomly. First, just like in cAS for TSP, we decide on the number of sampled positions using the distribution defined in Eq. 4. Then, the positions to sample are selected at random with a uniform distribution over all subsets of the specified size.

The second difference is in the meaning of the pheromone densities. In TSP, the pheromone density τ_{ij} corresponds to the desirability of including an edge (i, j) (from city i to city j) in a tour. On the other hand, in QAP, τ_{ij} corresponds to the desirability of putting a facility i at location j, similarly as in the prior work in ACO for QAP [3]. This affects both the generation of new assignments as well as the pheromone updates.

Finally, since in QAP we do not have a distance associated with each pheromone density, the probabilistic rule used for generating new permutations or assignments (see Eq. 5) sets $\beta = 0$ in the case of QAP, similarly as in MMAS [3]:

$$p_{i,j} = \frac{\tau_{i,j}^{\alpha}}{\sum_{k \in N(i)} \tau_{i,j}^{\alpha}} \tag{16}$$

All the remaining components of cAS for QAP remain the same as for TSP. Note that there is nothing to prevent us from using the cAS designed for TSP also on QAP; nonetheless, incorporating the prior knowledge about the problem structure of QAP allows cAS to better exploit problem regularities in the QAP problem domain.

5.3 Tested QAP instances

There are four basic types of QAP test instances [3, 29]: (1) randomly generated instances, (2) instances with a grid-based distance matrix, (3) real-life instances, and (4) real-life-like instances. In this study, we use instances that can be classified as real-life-like instances. All our test instances were developed by E. Taillard and they were obtained from QAPLIB [30, 31]. Specifically, we consider the following QAP instances: tai25b (n=25), tai30b (n=30), tai35b (n=35), and tai40b (n=40).

5.4 cAS on QAP without local search

This section analyzes performance of cAS, *non-*cAS and MMAS [3] on QAP. For all three algorithms, the maximum number of function evaluations was set as $E_{max} = n \times 800000$. Based on preliminary experiments, we set the size of the archive as $m = n \times 4$ where n is the problem size. We use $\rho = 0.9$ and $p_{best} = 0.005$. For cAS, we use $\gamma = 0.3$, unless explicitly indicated otherwise (*non-*cAS uses $\gamma = 1$). The code for MMAS is tuned by hand for the appropriate use of *global best* and *iteration best* in the pheromone updates in order to minimize error (maximize solution quality); smallest

Table 3 Performance of *c*AS and MMAS on QAP.

QAP	*c* AS				MMAS			
	c AS(γ=0.3)		*non-c* AS(γ=1)		MMAS+pts		MMAS	
	$Best_{avg}$	*Error (%)*	$Best_{avg}$	*Error (%)*	$Best_{avg}$	*Error (%)*	$Best_{avg}$	*Error (%)*
tai25b	344365211.3	**0.0028**	346471125.0	0.6143	350390715.6	1.7526	352569064.6	2.3852
tai30b	637536157.8	**0.0658**	671934840.6	5.4649	651604754.4	2.2739	651639592.6	2.2794
tai35b	284029941.4	**0.2522**	288833464.6	1.9477	290264755.1	2.4529	290319335.0	2.4721
tai40b	638166497.2	**0.1437**	662063702.4	3.8937	664608747.8	4.2931	665307422.9	4.4027

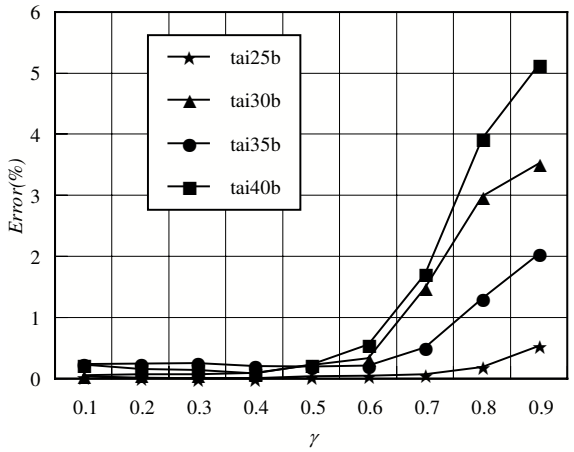

Fig. 9 Effect of γ on the errors obtained with *c*AS on QAP.

errors were obtained when the *global best* was applied every 5 iterations to the pheromone update. The *pts* strategy in MMAS was also tuned. For each QAP instance, 25 independent runs were performed.

The results on QAP are summarized in Table 3. The values in bold-face show the best performance for each instance. The results clearly indicate that *c*AS outperforms MMAS. We can also observe that *non-c*AS has similar performance as MMAS, similarly as for TSP.

5.5 Effect of γ on cAS performance on QAP

Similarly as for TSP, also for QAP we analyzed the effects of γ on performance of *c*AS. Fig. 9 shows the results. The results indicate that small values of γ lead to the best performance with respect to the errors obtained.

5.6 Measuring evolvability using pheromone entropy on QAP

Similarly as for TSP, also for QAP we can use the pheromone entropy $I(t)$ (see Eq. 9) or the normalized pheromone entropy $I_N(t)$ (see Eq. 14) to measure evolvability of cAS in order to estimate the likelihood of seeing further improvement. The only difference for QAP is that since edges that start and end in the same node are allowed, we need to include τ_{ii} in the computation of the entropy; thus, the entropy becomes

$$I(t) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}(t), \tag{17}$$

where $p_{ij}(t)$ is the normalized pheromone density at time t, defined as

$$p_{ij}(t) = \frac{\tau_{ij}(t)}{\sum_{j=1}^{n} \tau ij(t)}. \tag{18}$$

For QAP, both the upper and the lower bounds for $I(t)$ must be changed because of the different semantics of pheromone densities [22], yielding

$$High(I(t)) = I(0) = \log n, \tag{19}$$

and

$$Low(I(t)) = \log(r + n - 1) - \frac{r \log r}{r + n - 1}, \tag{20}$$

where $r = \tau_{max}/\tau_{min}$.

The convergence process including the value of the best-so-far solution and the normalized pheromone entropy for instances tai25b and tai30b is shown in figures 10 and 11. The results show that, similarly as in TSP, higher values of γ provide somewhat faster convergence but are also more likely to stagnate later in the run, whereas small values of γ lead to slower convergence but allow cAS to obtain a better solution in the end of the run. Once stagnation occurs in the runs with larger values of γ, the normalized pheromone entropy becomes relatively small.

6 Improving performance of *c*AS on QAP with local search

Here we study cAS with local search on QAP. In [3], MMAS is combined with two local searchers—robust tabu search (Ro-TS) developed by Taillard [29] and 2-OPT. Since Ro-TS is more powerful than 2-OPT, in this paper, we combined cAS with Ro-TS and compared the results obtained with those presented in [3]. The test instances are tai50b (n=50), tai60b (n=60), tai80b (n=80), tai100b (n=100), and tai150b (n=150).

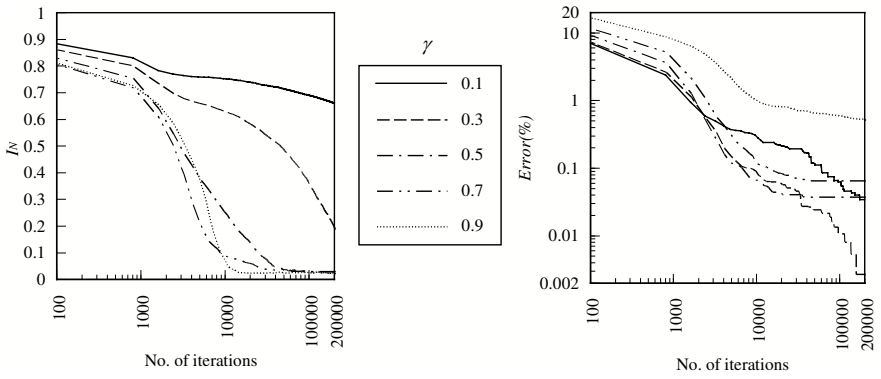

Fig. 10 Early stagnation and the normalized pheromone entropy for *c*AS on tai25b.

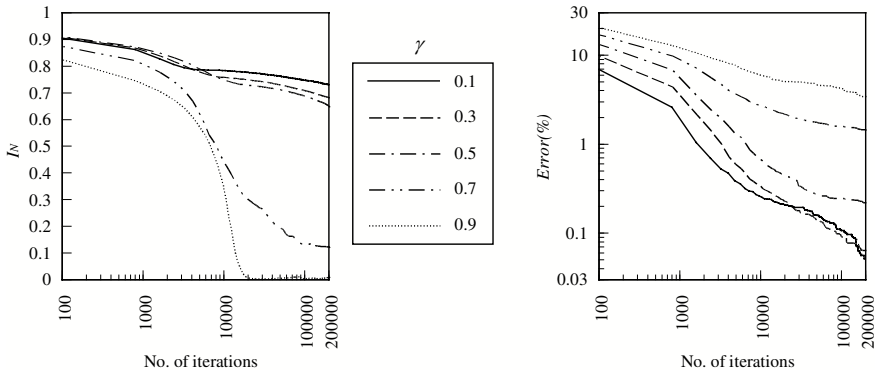

Fig. 11 Early stagnation and the normalized pheromone entropy for *c*AS on tai30b.

6.1 Summary of the results

Parameter settings and the methods of applying Ro-TS for *c*AS are the same as those used in [3]: $m = 5$, $\rho = 0.8$, and $p_{best} = 0.005$. For each candidate solution in *c*AS and MMAS, Ro-TS is allowed to run for $4n$ iterations and both *c*AS and MMAS are allowed to explore at most 250 candidate solutions in each run [3]; each run of *c*AS and MMAS is thus limited to $1000n$ steps of Ro-TS. For *c*AS, $\gamma = 0.8$ is used. To make these results compatible with those obtained with Ro-TS, Ro-TS alone was allowed to run also for $1000n$ iterations. We used the Ro-TS code available at [32], which we reimplemented using Java.

Table 4 summarizes the results. For a comparison, we show also the results obtained with other algorithms taken from [3], MMAS-TS, MMAS-2OPT, GH (genetic hybrid) [33], and Ro-TS alone. *c*AS again shows superior performance in terms of the quality of solutions found.

Table 4 Performance of cAS and MMAS with local search on QAP. The results for cAS are averaged over 25 independent runs, the remaining results are averaged over 10 independent runs.

QAP	c AS (γ=0.8)	MMAS-TS	MMAS-2OPT	GH	Ro-TS
tai40b	**0**	0.402	**0**	0.211	0.531
tai50b	**0.00113**	0.172	0.009	0.214	0.342
tai60b	**0.00091**	0.005	0.005	0.291	0.417
tai80b	**0.00445**	0.591	0.266	0.829	1.031
tai100b	**0.00155**	0.23	0.114	n.a.	0.512
tai150b	0.00727	n.a.	n.a.	n.a.	n.a.

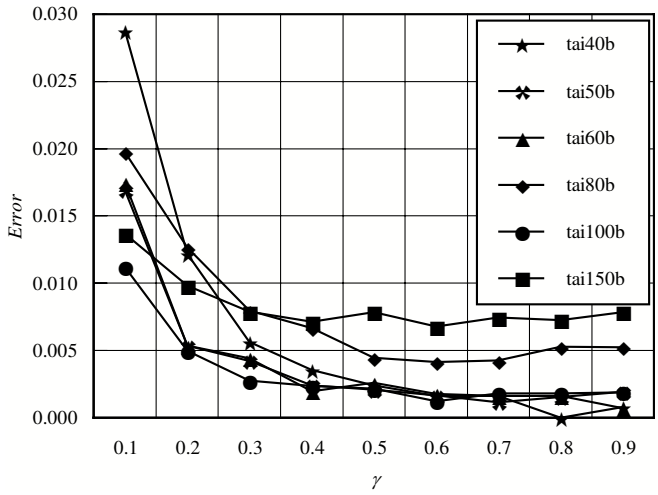

Fig. 12 Effect of γ on the errors obtained with cAS with local search on QAP.

6.2 Effect of γ on cAS performance on QAP

To see the effect of γ value, we show the change of *Error* for various γ of values, which are from 0.1 to 0.9 with step size 0.1, in Fig. 12. As we saw in Fig. 9 in Subsection 4.2, cAS without local search works well for the smaller values of γ (in the range of [0.1, 0.5]). In contrast to this, we can see that cAS with local search works well for the larger values of γ (in the range of [0.5, 0.8]).

The convergence process and the pheromone entropy for cAS with Ro-TS on the QAP instance tai100b is shown in Fig. 13. The results show that for cAS with Ro-TS, early stagnation has not been observed as is also supported by the relatively high values of the pheromone entropy even late in the run. Analogical performance was observed for other tested QAP instances.

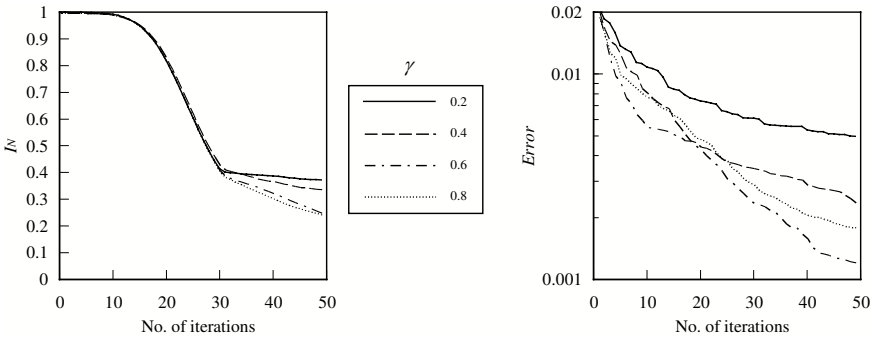

Fig. 13 Convergence and the normalized pheromone entropy for *c*AS with local search on the QAP instance tai100b.

7 Parallel *c*AS

Parallelization is yet another popular efficiency enhancement technique, which is often used for solving extremely difficult and high-dimensional problems [34–37]. There are various approaches to parallelizing ACO and other evolutionary algorithms. One approach is to use the island evolutionary model where the colony is distributed to several isolated processing units, which communicate and exchange information in some way at given intervals [37]. Another approach is to distribute the procedures for generating and evaluating solutions using the master-slave architecture [36]. In this paper, we focus on the latter approach and propose parallelization of solution evaluation and local search (Ro-TS), which is the computational bottleneck of *c*AS with Ro-TS (see section 7.2). Our primary goal is to maximize the *c*AS speedups obtained.

7.1 The basic framework of parallel cAS

All algorithms discussed in the paper were implemented in Java. Java has many classes for programming in network environments; typical examples are RMI and Jini [38]. However, for the sake of simplicity and portability, we decided to use a different approach and we based our implementation of parallel *c*AS on Java applets. This enables us to use all computers connected to the network assuming that they have a standard web browser compatible with Java Runtime Environment without having to install any additional software on them. Communication between the server and clients is performed by exchanging objects using the *serializable interface*. Only the server is required to contain the main application written in Java.

After each of the clients connects to the server via the web browser interface and the socket connection between the server and the clients is established, the server starts its execution (see Fig. 14). The work can be distributed in many different ways; here we only distribute the methods that

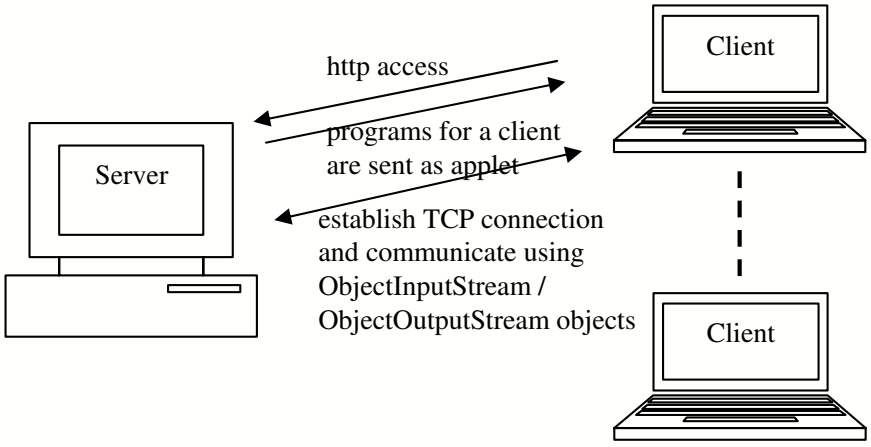

Fig. 14 The overall structure of the parallel *c*AS (*p*-*c*AS).

evaluate candidate solutions and run a local search technique to improve them. All data files used by clients, such as the definition of problem instance being solved, are converted to classes so that they are accessible from the clients using Java.

7.2 Time complexity analysis of cAS with Ro-TS on QAP

In order to design an efficient parallel version of *c*AS, it is important to first analyze time complexity of the different components of *c*AS. That is why we recorded time spent in different parts of *c*AS with Ro-TS using the same experimental setup as we used in Sect. 6. All experiments were done on a computer with two Opteron 280 (2.4GHz, Socket940) processors with 2GB main memory. The operating system was a 32-bit Windows XP, and we used Java2 (j2sdk1.4.2_13).

The results are summarized in Table 5. The results show that more than 99% of computation time in *c*AS with Ro-TS is used in Ro-TS. Therefore, to maximize the speedups, it's important to focus on Ro-TS.

7.3 Description of parallel cAS

Parallel *c*AS discussed in this paper uses the standard master-slave parallel architecture. Most of the work is done on the master, which executes all components of *c*AS except for the solution evaluation and the Ro-TS, which is applied to each generated solution. Instead, after generating the new solutions, these are distributed between the clients and the master. The clients and the master evaluate these solutions and apply Ro-TS to them,

Table 5 Computational time of *c*AS with Ro-TS on QAP (in milliseconds).

QAP	sampling based on τ_{ij} with cunning	applying Ro-TS	updating of τ_{ij}	other (for statics, etc.)	TOTAL
tia40b	3.8	6833.8	1.3	12.3	6851.2
(%)	0.06%	99.75%	0.02%	0.18%	100.00%
tia50b	4.9	13465.8	2.3	13.3	13486.2
(%)	0.04%	99.85%	0.02%	0.10%	100.00%
tia60b	7.0	23448.7	3.0	15.7	23474.4
(%)	0.03%	99.89%	0.01%	0.07%	100.00%
tia80b	10.6	56688.7	5.9	19.9	56725.0
(%)	0.02%	99.94%	0.01%	0.04%	100.00%
tia100b	14.8	114411.7	9.0	21.4	114456.9
(%)	0.01%	99.96%	0.01%	0.02%	100.00%
tai150b	29.9	422079.8	19.9	46.1	422175.6
(%)	0.01%	99.98%	0.00%	0.01%	100.00%

and then send the results back to the master. The master collects the results and proceeds with the update of the archive and the pheromone updates.

To use the computational resources most efficiently, it is important to ensure that all processors process approximately the same number of ants (the processor with most ants is most likely going to become the computational bottleneck). Ideally, the number of ants is an integer multiple of the number of processing units (number of clients plus one); in this case, the number of ants processed on each processing unit is the same. In the worst case, some processors will have one additional ant compared to others.

The division of labor between the server and the clients is visualized in Fig. 14. Note that the application of parallel *c*AS is not limited to QAP; the same idea can be used for parallelizing *c*AS in other problem domains, such as TSP.

7.4 Experimental setup

We used two Opteron-based machines (A and B) and each of these machines had two dual-core processors. Thus, each machine had four processing units. The machines were connected with a 1000BASE-T switch. The machine A acted as the server; all client processes were located at machine B.

We assigned server functions to machine A and client functions to machine B. In machine A, we installed an Apache [39] HTTP server. Since machine B would execute only the clients, we only needed a standard Java-compatible web browser on this machine. Since the experiments presented earlier in Table 4 used only 5 ants in the archive, all processing units process exactly one ant. All parameters were set the same as in the experiments presented earlier in Table 4.

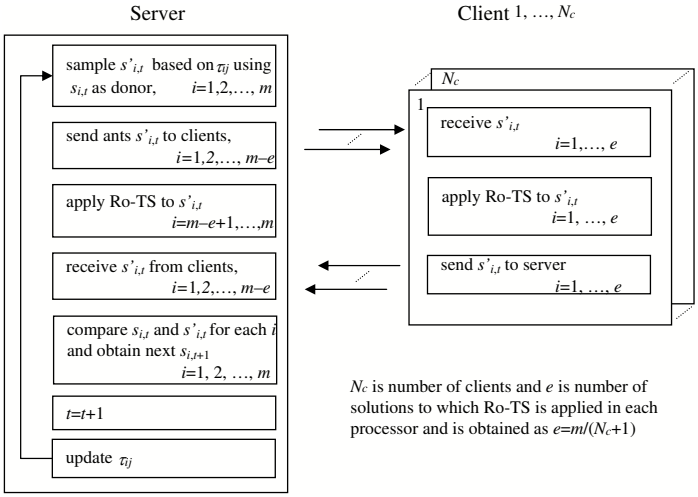

Fig. 15 Basic outline of the parallel cAS (p-cAS).

7.5 Speedups obtained with parallel cAS on QAP

Fig. 16 summarizes the results obtained with parallel cAS (averaged over the 25 independent runs). Here, *speedup* indicates the ratio of the sequential running time and the parallel running time; therefore, *speedup* represents the factor by which the computational time required by cAS decreases. The higher the speedup, the shorter the execution time.

Ideally, if there was no communication overhead, the speedup should be about five because vast majority of the overall running time is spent in Ro-TS, which is equally distributed to the five processing units. However, as we can see in Fig. 16, due to the communication overhead, the actual speedups are smaller. In fact, for the smallest problems, the communication overhead overshadows the benefits of parallelization, leading to speedups smaller than one (that means that the overall computational time in fact *increased*). However, as the problem size increases, the time spent in Ro-TS grows faster than the communication overhead, leading to the speedup of 4.1 for the largest problem (tai150b). Therefore, for big problems, the speedups obtained can be expected to be significant.

8 Summary and Conclusions

This paper described the cunning ant system (cAS), which can be applied to optimization problems defined over permutations, such as the traveling salesman problem and the quadratic assignment problem. The cunning ant system maintains an *archive* of promising solutions found so far. New candidate solutions are generated by using a combination of the solutions in

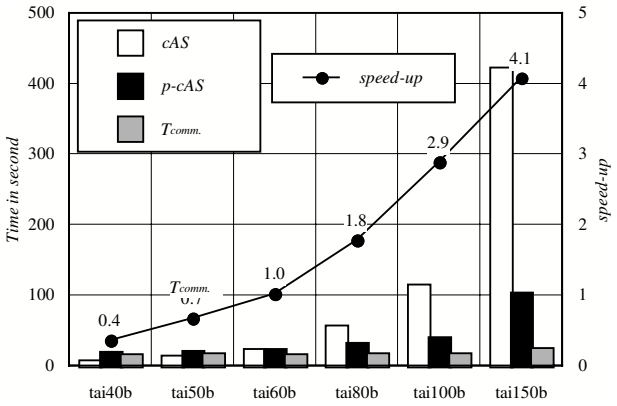

Fig. 16 Performance of *p*-*c*AS and the speedups obtained on QAP.

the archive and the pheromone densities, which are continuously updated based on the archive.

Performance of *c*AS was tested on a number of problem instances of two important classes of optimization problems: (1) the traveling salesman problem and (2) the quadratic assignment problem. In both problem classes, *c*AS outperformed other optimization techniques included in the comparison.

The paper also presented two efficiency enhancement techniques, which can be used to further improve the efficiency of *c*AS. First, we proposed hybrid techniques created by combining *c*AS with efficient local searchers for TSP and QAP, and then suggested a parallel implementation of *c*AS. Both efficiency enhancement techniques were shown to significantly improve the efficiency of *c*AS.

The paper showed that *c*AS provides competitive performance on a broad range of problem instances and that this approach to optimization of problems defined over permutations holds a big promise.

There are several interesting topics for future work in this area. First of all, this paper showed that various types of regularities (nearest-neighbor dependencies and absolute-position dependencies) typically observed in permutation-based problems can be directly exploited within the ACO framework using *c*AS; are the two variants of *c*AS discussed in this paper sufficient to solve all standard permutation-based problems and if not, what other types of regularities can be incorporated into *c*AS to extend this technique to other important problem classes? Second, we discussed the master-slave parallel architecture, which allowed us to obtain significant speedups for difficult permutation-based problems; what are the benefits of other approaches to parallelizing *c*AS, such as the popular island model? Finally, how will *c*AS perform on other important TSP and QAP benchmarks and real-world problems?

Acknowledgments

This project was sponsored by the Ministry of Education, Culture, Sports, Science and Technology of Japan under Grant-in-Aid for Scientific Research No. 19500199, by the National Science Foundation under CAREER grant ECS-0547013, by the Air Force Office of Scientific Research under grant FA9550-06-1-0096, and by the University of Missouri in St. Louis through the Research Award Program and the High Performance Computing Collaboratory sponsored by Information Technology Services. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF, AFOSR, or the U.S. Government.

References

1. Dorigo, M., Maniezzo, V., Colorni, A.: Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics - 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 Transactions on Evolutionary Computation **1**(1) (1997) 53–66
3. Stützle, T., Hoos, H.: Max-min ant system. Future Generation Computer Systems **16**(9) (2000) 889–914
4. 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
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. Gambardella, L.M., Taillard, É.D., Dorigo, M.: Ant colonies for the QAP. Technical Report IDSIA-4-97, Swiss Institute for Artificial Intelligence (IDSIA), Lugano, Switzerland (1997)
7. Maniezzo, V., Colorni, A.: The ant system applied to the quadratic assignment problem. IEEE Transactions on Knowledge and Data Engineering **11**(5) (1999) 769–778
8. Socha, K., Dorigo, M.: Ant colony optimization for continuous domains. European Journal of Operational Research (2007) In press.
9. Pourtakdoust, S.H., Nobahari, H.: An extension of ant colony system to continuous optimization problems. Proc. of Fourth Int. Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2004) (2004) 294–301
10. Dorigo, M.: Optimization, Learning and Natural Algorithms (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy (1992)
11. Acan, A.: An external memory implementation in ant colony optimization. Proc. of the 4th International Workshop on Ant Algorithms and Swarm Intelligence (ANTS-2004) (2004) 73–84
12. Acan, A.: An external partial permutations memory for ant colony optimization. Proc. of the 5th European Conf. on Evolutionary Computation in Combinatorial Optimization (2005) 1–11

13. Wiesemann, W., Stützle, T.: An experimental study for the the quadratic assignment problem. Proc. of the 5th Int. Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2006) (2006) 179–190
14. 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)
15. 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
16. Biggs, N., LLoyd, E., Wilson, R.: Graph Theory 1736–1936. Clarendon Press, Oxford (1976)
17. TSPLIB. (`http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`)
18. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT press, MA (2004)
19. Gambardella, L.M., Dorigo, M.: Solving symmetric and asymmetric tsps by ant colonies. Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC'96) (1996) 622–627
20. Wagner, A.: Robustness and Evolvability in Living Systems. Princeton University Press (2005)
21. Tsutsui, S., Lichi, L.: Convergence analysis of the cunning ant system using entropy measure. Submitted to SMC conference. (2007)
22. Tsutsui, S., Lichi, L.: Cunning ant system for quadratic assignment problem with local search and parallelization. Submitted to EA-2007. (2007)
23. Goldberg, D.E.: The design of innovation: Lessons from and for competent genetic algorithms. Kluwer Academic Publishers (2002)
24. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling salesman problem. Operations Research **21** (1973) 498–516
25. Applegate, D., Cook, W., Rohe, A.: Chained Lin-Kernighan for large traveling salesman problems. INFORMS J. on Computing **15** (2003) 82–92
26. 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
27. Applegate, D., et al.: ANSI C code as gzipped tar file, Concorde tsp solver (2006) `http://www.tsp.gatech.edu/concorde.html`.
28. Koopmans, T.C., Beckmann, M.J.: Assignment problems and the location of economic activities. Econometrica **25** (1957) 53–76
29. Taillard, E.: Robust tabu search for the quadratic assignment problem. Parallel Computing **17** (1991) 443–455
30. Burkard, R.E., Karisch, S., Rendl, F.: QAPLIB – A quadratic assignment problem library. ejor **55** (1991) 115–119 www.opt.math.tu-graz.ac.at/qaplib/.
31. Burkard, R., Karisch, S., Rendl, F.: QAPLIB – A quadratic assignment problem library. Journal of Global Optimization **10** (1997) 391–403
32. Taillard, E.: (Robust tabu search implementation) `http://mistic.heig-vd.ch/taillard/`.
33. Fleurent, C., Ferland, J.: Genetic hybrids for the quadratic assignment problem. DIMACS Series in Mathematics and Theoretical Computer Science **16** (1994) 190–206
34. Cantu-Paz, E.: Efficient and Accurate Parallel Genetic Algorithm. Kluwer Academic Publishers, Boston (2000)
35. Tsutsui, S., Fujimoto, Y., Ghosh, A.: Forking GAs: GAs with search space division schemes. Evolutionary Computation **5**(1) (1997) 61–80

36. Randall, M., Lewis, A.: A parallel implementation of ant colony optimization. Journal of Parallel and Distributed Computing **62**(9) (2002) 1421–1432
37. Manfrin, M., Birattari, M., Stützle, T., Dorigo, M.: Parallel ant colony optimization for the traveling salesman problems. Proc. of the 5th Int. Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2006) (2006) 224–234
38. Sun Microsystems, Inc.: (Java 2 Platform, Standard Edition, v1.4.2 at API Specification) `http://java.sun.com/j2se/1.4.2/docs/api/`.
39. Apache Software Foundation: (Apache HTTP server project) `http://httpd.apache.org/`.