Cunning Ant System for Quadratic Assignment Problem with Local Search and Parallelization

Shigeyoshi Tsutsui

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

Abstract. The previously proposed *cunning* ant system (*c*AS), a variant of the ACO algorithm, worked well on the TSP and the results showed that the *c*AS could be one of the most promising ACO algorithms. In this paper, we apply *c*AS to solving QAP. We focus our main attention on the effects of applying local search and parallelization of the *c*AS. Results show promising performance of *c*AS on QAP.

1 Introduction

In a previous paper [1,2], we have proposed a variant of the ACO algorithm called the *cunning* Ant System (*c*AS) and evaluated it using TSP which is a typical NP-hard optimization problem. The results showed that the *c*AS could be one of the most promising ACO algorithms. In this paper, we apply *c*AS to solving the quadratic assignment problem (QAP). The QAP is also an NP-hard optimization problem and it is considered one of the hardest optimization problems [3,4]. The QAP is also a good set of problems for testing the capabilities of solving combinatorial optimization problems.

There are many studies on solving QAP with ACO showing better results than with other meta-heuristics. These studies are summarized in [5]. Typical examples of ACO algorithms for the QAP are AS-QAP, MMAS-QAP, and ANTS-QAP. Among these, it is reported that MMAS-QAP [3] is the best performing algorithm [5].

We performed a preliminary study which applied *c*AS to solving QAP in [6]. In this paper, we apply *c*AS to solving QAP and compare the performance with the performance of MMAS [3]. We also discuss an approach for parallelization of the *c*AS for QAP.

In the remainder of this paper, Section 2 gives a brief overview of *c*AS when it is applied in TSP. Then, Section 3 describes how the solutions with *c*AS for the QAP are constructed. In Section 4, we provide an empirical analysis of the *c*AS and compare the results with MMAS. In Section 5, we study the use of a kind of parallelization of *c*AS, with the aim of achieving faster execution of the algorithm in a network environment. Finally, Section 6 concludes this paper.

A. Ghosh, R.K. De, and S.K. Pal (Eds.): PReMI 2007, LNCS 4815, pp. 269–278, 2007.

2 A Brief Overview of *c*AS

*c*AS [1,2] introduced two important schemes. One is a scheme to use partial solutions which we call *cunning*. The other is to use the colony model, dividing colonies into units. Using partial solutions to seed solution construction in the ACO can be found in [7,8,9] with other frameworks. The agent introduced in *c*AS is called *cunning ant* (*c-ant*). The *c-ant* differs from traditional ants in its 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. An agent from whom a partial solution has been borrowed by a *c-ant* is called a *donor ant* (*d-ant*).

We use a colony model which consists of m units [1,2]. Each unit consists of only one $ant^*_{k,t}$ ($k = 1, 2, \ldots, m$). 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.

Pheromone density $\tau_{ij}(t)$ is then updated with $ant^*_{k,t}$ (k=1, 2, ..., m) and $\tau_{ij}(t+1)$ is obtained as:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij} + \sum\nolimits_{k=1}^{m} \Delta^* \tau^k_{ij}(t), \tag{1}$$

$$\Delta^* \tau^k_{ij}(t) = 1/C^*_{k,t} : \text{ if } (i,j) \in ant^*_{k,t},\ 0 : \text{ otherwise}, \tag{2}$$

where the parameter ρ ($0 \leq \rho < 1$) is the trail persistence (thus, $1-\rho$ models the evaporation), $\Delta^* \tau^k_{ij}(t)$ is the amount of pheromone $ant^*_{k,t}$ puts on the edge it has used in its tour, and $C^*_{k,t}$ is the fitness of $ant^*_{k,t}$.

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

$$\tau_{\max}(t) = \frac{1}{1-\rho} \times \sum\nolimits_{k=1}^{m} \frac{1}{C^*_{k,t}}, \tag{3}$$

$$\tau_{\min}(t) = \frac{\tau_{\max} \cdot (1 - \sqrt[n]{p_{best}})}{(n/2 - 1) \cdot \sqrt[n]{p_{best}}}, \tag{4}$$

where p_{best} is a control parameter introduced in MMAS [3].

3 Cunning Ant System for QAP

The QAP is a problem in which a set of facilities or units are assigned to a set of locations and can be stated as a problem to find permutations which minimize

$$f(\phi) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{ij} b_{\phi(i)\phi(j)}, \tag{5}$$

where $A = (a_{ij})$ and $B = (b_{ij})$ are two $n \times n$ matrices and ϕ is a permutation of $\{0, 1, \ldots, n–1\}$. Matrix A is a distance matrix between locations i and j, and B is the flow between facilities r and s. Thus, the goal of the QAP is to place the facilities on locations in such a way that the sum of the products between flows and distances are minimized.

3.1 The *c-ant* for QAP

The *c-ant* in QAP acts in a slightly different manner than a *c-ant* in TSP. In TSP, pheromone trails $\tau_{ij}(t)$ are defined on each edge between city i and j. On the other hand, the pheromone trails $\tau_{ij}(t)$ in the QAP application correspond to the desirability of assigning a facility i to a location j [3]. In this paper, we use this approach for cAS on QAP. Fig. 1 shows how the *c-ant* acts in QAP.

In this example, the *c-ant* borrows part of the node values at location 0, 2, and 4. The *c-ant* constructs the remainder of the node values for location 1 and 3 according to the following probability:

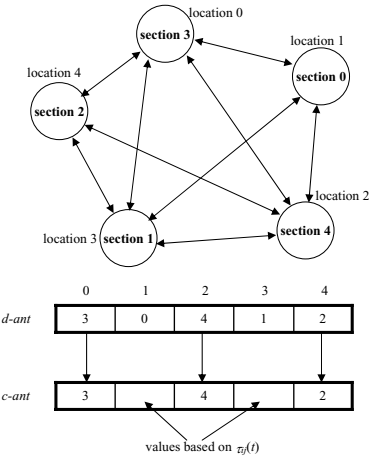

Fig. 1. *c-ant* and *d-ant* in QAP

$$p_{ij}(t) = \frac{\tau_{ij}(t)}{\sum_{k \in N(i)} \tau_{ik}}, \qquad (6)$$

where $N(i)$ is the set of still unassigned facilities. Using *c-ant* in this way, we can prevent premature stagnation the of search, because only a part of the nodes in a string are newly generated, and this can prevent over exploitation caused by strong positive feedback to $\tau_{ij}(t)$ as we observed in cAS in [1,2]. The colony model of cAS for QAP is the same as was used in [1,2] for TSP.

3.2 Sampling Methods

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 $l_c = n–l_s$. Following cAS in TSP, we use the control parameter γ which define $E(l_s)$ (the average of l_s) by $E(l_s) = n \times \gamma$ and use the following probability density function $f_s(l)$ used in [1,2] as

$$f_s(l) = \begin{cases} \frac{1-\gamma}{n\gamma} \left(1 - \frac{l}{n}\right)^{\frac{1-2\gamma}{\gamma}} & \text{for } 0 < \gamma \leq 0.5, \\ \frac{\gamma}{n(1-\gamma)} \left(\frac{l}{n}\right)^{\frac{2\gamma-1}{1-\gamma}} & \text{for } 0.5 < \gamma < 1. \end{cases} \qquad (7)$$

In cAS for TSP, nodes in continuous positions of *d-ant* are copied to *c-ant*, because the partial solutions of *d-ant* are represented by nodes in continuous positions. However, in QAP there is no such constraint and it is not necessary

for nodes, which are copied from *d-ant* or sampled according to $\tau_{ij}(t)$, to be in continuous positions. Thus, in creating a new *c-ant* in QAP, nodes at some positions are copied and others are sampled with a random sequence of positions as follows: The number of nodes to be sampled l_s is generated by Eq. 7 with a given γ value. Then we copy the number of nodes, $l_c = n–l_s$, from *d-ant* at random positions and sample the number of remaining nodes, l_s, according to Eq. 6 with random sequence.

4 Experiments

QAP test instances in QAPLIB [10] can be classified into i) randomly generated instances, ii) grid-based distance matrix, iii) real-life instances, and iv) real-life-like instances [11,3]. In this section, we evaluate *c*AS on the QAP using QAPLIB instances which were used in [3] and compare the performance with MMAS.

4.1 Performance of *c*AS on QAP Without Local Search

Here, we see the performance of *c*AS without local search using relativlely small instances showned in Table 1. The comparison with MMAS was performed on the same number of solution constructions $E_{max} = n\times800{,}000$. For number of units (or ants for MMAS) $m = n\times4$ is used. ρ value of 0.9 and p_{best} value of 0.005 are used for both *c*AS and MMAS. 25 runs were performed.

Table 1 summarizes the results. The values in the table represent the deviation from the optimum value by *Error* (%) $((f(\phi) - best)/best \times 100)$. The results of *c*AS are with γ value of 0.3. The code for MMAS is implemented by us and tuned for the appropriate use of *global best* and *iteration best* in the pheromone update so as to get the smallest values of *Error*. We got the smallest value when the *global best* was applied every 5 iterations to the pheromone update in MMAS. The *pts* strategy [3] in MMAS was also tuned.

The values in bold-face show the best performance for each instance. From this table, we can see that *c*AS has good performance.

Table 1. Results without local search. *Error*(%) is average over 25 independent runs.

QAP class	QAP instance	*c*AS (γ=0.3)	MMAS	
			MMAS+pts	MMAS
i	tai20a	**1.006**	2.996	3.140
	tai25a	**1.566**	3.217	3.380
	tai30a	**1.843**	3.004	2.758
	tai35a	**2.194**	3.690	3.600
ii	nug30	**0.455**	1.675	1.962
iii	kra30a	**1.147**	3.963	4.014
	kra30b	**0.447**	2.736	2.836
iv	tai20b	**0.000**	0.348	0.388
	tai25b	**0.003**	1.753	2.385
	tai30b	**0.066**	2.274	2.279
	tai35b	**0.252**	2.453	2.472

4.2 The Effect of γ Values

Table 1 shows *Error* for γ = 0.3. Fig. 2 shows the variations of *Error* for various γ values on tai30aCnug30Ckra30b, and tai30b. Here, γ values were varied starting from 0.1 to 0.9 with step 0.1. From this figure, we can see the effectiveness of using *c-ant*; i.e., with the smaller values of γ (in the range of [0.1, 0.5]), the better values in *Error* are observed as was the case with *c*AS on TSP in [1,2].

4.3 Analysis of the Convergence Process of *c*AS

As we discussed in Section 2 and Subsection 4.2, the cunning action can be expected to prevent premature stagnation the of search, because only a part of the nodes in a solution are newly generated, and this prevent over exploitation caused by strong positive feedback to $\tau_{ij}(t)$. In this subsection, we analyze the convergence process using *Entropy* of pheromone density $\tau_{ij}(t)$ to measure the diversity of the system.

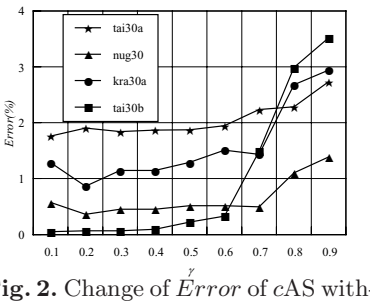

Fig. 2. Change of $\overline{Error}^{\gamma}$ of *c*AS without local search for various γ

Definition of entropy of pheromone density. We define $I(t)$, entropy of pheromone density $\tau_{ij}(t)$, as follows:

$$I(t) = -\frac{1}{n}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} p_{ij}(t)\log p_{ij}(t), \tag{8}$$

where $p_{ij}(t)$ is defined as

$$p_{ij}(t) = \frac{\tau_{ij}(t)}{\sum_{j=0}^{n-1}\tau_{ij}(t)}. \tag{9}$$

The upper bound of $I(t)$ is obtained when all elements of $\tau_{ij}(t)$ have the same values as found during the initialization stage (t=0). This value is calculated as

$$\overline{I} = \log(n). \tag{10}$$

To calculate the lower bound of $I(t)$, let's consider an extreme case in which all strings have the same set of node values and pheromones have been distributed across the set. If this iteration continues for a long time, all elements of $\tau_{ij}(t)$ converge to τ_{min} or τ_{max}. The lower bound of the *entropy* $I(t)$ is obtained in these situations and can be calculated as Eq. 11 as follows:

$$\underline{I} = \log(r+n-1) - \frac{r\log(r)}{r+n-1} \tag{11}$$

where r=τ_{max}/τ_{min}. In the following analysis, we use the normalized entropy $I_N(t)$ which is defined with $I(t)$, $\overline{I}$, and $\underline{I}$ as

$$I_N(t) = \frac{I(t)-\underline{I}}{\overline{I}-\underline{I}} \tag{12}$$

Then, $I_N(t)$ takes values in [0.0, 1.0].

Analysis of the convergence process. Here we show the convergence processes for tai25b and tai30b in Fig. 3. In the figure, the left shows the change in *Error* (%) and the right shows the change in $I_N(t)$. Values in the figure show averaged values over 25 independent runs. On tai25 with γ values of 0.5, 0.7, and 0.9 in (a), we can see that I_N converges around 80000, 40000, and 20000 iterations, respectively. These iterations coincide with the iterations where stagnations in *Error* occur. With γ value of 0.3, the value of I_N gradually decreases and the search continues with less stagnation. With γ value of 0.1, I_N keeps larger values until the end of run, resulting in slow convergence in *Error*. Similar results for tai30b are observed, although their values in detail are different from tai25b.

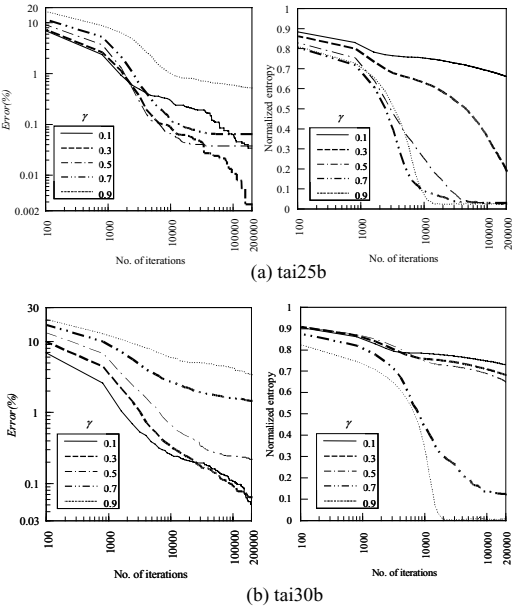

Fig. 3. Convergence processes of tai25b and tai30b without local search

From this convergence process analysis using the entropy measure, we can see the effectiveness of the cunning scheme with smaller values of γ. That is, on average, taking the rate of $(1-\gamma)$ partial solution from existing solutions, and having the rate of γ partial solution being generated anew from the pheromone density can maintain diversity of the system, resulting in good balance between exploration and exploitation in the search. However, with extreme smaller values of γ, i.e., $\gamma \leq 0.1$, the search processes becomes much slower, though the diversity of pheromone density can be maintained.

4.4 Performance of *c*AS with Local Search

Here we study *c*AS with a local search on QAP. In [3], MMAS is combined with two local searches, i.e., Robust Taboo search algorithm (Ro-TS) developed by Taillard [11] and 2OPT. In this paper, we combined *c*AS with Ro-TS (*c*AS-TS) and compare the results with results described in [3].

Parameter settings and the methods of applying *c*AS with Ro-TS (*c*As-TS) are the same as were used for MMAS with To-TS (MMAS-TS) in [3] as follows: m value of 5, ρ value of 0.8, and p_{best} value of 0.005. 250 times, short Ro-TS runs of length $4n$ were applied. This setting was designed in [3] so that the computational time is the same as the Ro-TS carried out alone in which $1000 \times n$ iterations was allowed. We used the Ro-TS code which is available at [12], though

the code, which is originally written in C, was rewritten in Java since our *c*AS code is written in Java.

Table 2 summarizes the results. For comparison, we show results of other algorithms, i.e., MMAS-TS, MMAS-2OPT, GH (Gentic Hybrid), HAS (Hybrid Ant System), and Ro-TS. These are taken from [3]. Results of *c*AS-TS is for γ=0.4 and 0.8. First, we compare *c*AS-TS with MMAS-TS. In this comparison, we showed the better values in bold-face, and showed the best performing values in bold-face with an under line.

Table 2. Results of *c*AS in *Error* (%) with local search. Results except for *c*AS are average over 10 runs.

QAP class	QAP	*c* AS-TS (γ=0.4)	*c* AS-TS (γ=0.8)	MMAS-TS	MMAS-2OPT	HAS-QAP	GH	Ro-TS
i	tai35a	**0.582**	**0.572**	0.715	1.128	1.762	0.698	0.589
	tai40a	**0.726**	0.793	0.794	1.509	1.989	0.884	0.990
	tai50a	**1.051**	1.190	1.060	1.795	2.800	1.049	1.125
	tai60a	**1.059**	1.289	1.137	1.882	3.070	1.159	1.203
	tai80a	**0.740**	1.029	0.836	1.402	2.689	0.796	0.900
ii	sko42	**0.005**	0.008	0.032	0.051	0.076	**0.003**	0.025
	sko49	**0.044**	0.062	0.068	0.115	0.141	**0.040**	0.076
	sko56	**0.055**	**0.065**	0.075	0.098	0.101	0.060	0.088
	sko64	0.077	**0.035**	0.071	0.099	0.129	0.092	0.071
	sko72	0.126	0.091	**0.090**	0.172	0.277	0.143	0.146
	sko81	0.105	0.105	**0.062**	0.124	0.144	0.136	0.136
	sko90	**0.104**	0.168	0.114	0.140	0.231	0.196	0.128
iii	ste36a	0.139	0.118	**0.061**	0.126	n.a.	n.a.	0.155
	ste36b	**0**	**0**	**0**	**0**	n.a.	n.a.	0.081
iv	tai35b	0.098	**0**	0.051	**0**	0.026	0.107	0.064
	tai40b	0.403	**0**	0.402	**0**	0.000	0.211	0.531
	tai50b	0.183	**0.113**	0.172	**0.009**	0.192	0.214	0.342
	tai60b	0.280	0.091	**0.005**	**0.005**	0.048	0.291	0.417
	tai80b	0.716	**0.445**	0.591	**0.266**	0.667	0.829	1.031
	tai100b	0.263	**0.155**	0.230	**0.114**	n.a.	n.a.	0.512

For instances in class i), *c*AS-TS performed better than MMAS-TS and and was the best performer. Here note that *c*AS-TS with $\gamma = 0.4$ showed better performance than *c*AS-TS with $\gamma = 0.8$. For instances in class ii), *c*AS-TS showed better performance than MMAS-TS except for with sko72 and sko81. Note here that GH showed the best values among all algorithms, but the performance differences between GH and *c*AS-TS were very small.

For instances in class iii), MMAS-TS performe better than *c*AS-TS on ste36a, and *Error*=0 on ste36b for both *c*AS-TS and MMAS-TS. For instances in class iv), with all instances except tai60b, *c*AS-TS showed better *Error* values than MMAS-TS. Note here that for all instances in this class, MMAS-2OPT has the best *Error* values among the algorithms.

In comparison between *c*AS-TS and MMAS-TS, *c*AS-TS outperforms MMAS-TS on 15 instances and MMAS-TS outperforms *c*AS-TS on 4 instances. Thus, *c*AS-TS has relatively better performance than MMAS-TS. However, the performance differences between *c*AS-TS and MMAS-TS were not as large as those we saw in Table 1 where no local search is applied. This is because the effect of local searches become more dominant.

5 Parallelization of *c*AS (*p*-*c*AS)

Parallelization of evolutionary algorithms including ACO is a well-known and popular approach [13,14,15]. There are two main reasons for using parallelization: (i) given a fixed time to search, to increase the quality of the solutions found

within that time; (ii) given a fixed solution quality, to reduce the time to find a solution not worse than that quality [15]. In this study, we ran a *parallel* cAS (p-cAS) exploiting the second reason, i.e., aiming to reduce the time to find solutions which are of the same quality as those found with a single processor.

All of our code for evolutionary computation research are written in Java, including ACO algorithms. Java has many *classes* of programming for network environments. Typical examples are RMI and Jini [16]. In our implementation of p-cAS, we used *Java applet* scheme to send the program to client computers. This enables us to use all computers in the network which have a web browser with Java runtime. Communication between the server and clients is performed by exchanging *objects* with the *Serializable interface*. The server program runs as a Java application.

5.1 Load Sharing p-cAS on QAP

In the ACO framework, the most popular parallel architecture is the *island model* in which multiple sub-colonies are run in parallel on distributed computers exchanging information among them periodically [15]. The main priority of the *island model* is placed on improving the solution quality. In contrast, our main priority is to reduce computational time using the *load sharing model* (*load sharing* p-cAS).

Table 3. Comptational time of cAS with Ro-TS in millisecond

QAP	sampling with τ_{ij}	applying Ro-TS	updating of τ_{ij}	other	TOTAL (ms)
tia40b	3.8	6833.8	1.3	12.3	6851.2
(%)	0.1%	99.7%	0.0%	0.2%	100.0%
tia50b	4.9	13465.8	2.3	13.3	13486.2
(%)	0.0%	99.8%	0.0%	0.1%	100.0%
tia60b	7.0	23448.7	3.0	15.7	23474.4
(%)	0.0%	99.9%	0.0%	0.1%	100.0%
tia80b	10.6	56688.7	5.9	19.9	56725.0
(%)	0.0%	99.9%	0.0%	0.0%	100.0%
tia100b	14.8	114411.7	9.0	21.4	114456.9
(%)	0.0%	100.0%	0.0%	0.0%	100.0%

Table 3 shows the computation times on QAP in cAS with Ro-TS which were performed in Section 4.4. The machine we used had two Opteron 280 (2.4GHz, Socket940) processors with 2GB main memory. The OS was 32-bit WindowsXP. Java2 (j2sdk1.4.2_13) was installed. From this table, we can see that more than 99% of computation time is used for Ro-TS. Therefore we distribute the calculation for local search over computers in the network. Fig. 4 shows the functions of the server and clients. When we use m ants, local searches for m ants are distributed over the server and m–1 clients.

Experimental conditions for the *load sharing* p-cAS in this research are as follows: We used two Opteron-based machines, say machine A and machine B, each which has the structure described above and thus each machine has 4 processing units. The machines A and B are connected via a 1000BASE-T switching hub. We assigned server functions to machine A and client functions to machine B. In machine A, we installed an Apache [17] http server. We assigned four clients so that the logical experimental conditions are the same as the experiments in Section 5 with a single machine. To do so, we ran 4 independent browser processes to access the server. The experiments were performed with 25 independent runs

and the time to complete the computation was measured. We used γ value of 0.8. With this scheme, among 5 ants, local search for one ant is performed by server machine A, and local searches for the other 4 ants are performed by client machine B.

Fig. 5 summarizes the results on the computation time. The results are averaged over 25 runs. Here, *gain* indicates (run time of cAS) / (run time of p-cAS). If there is no communication overhead, gain should be five. However, as we can see in Fig. 5 there is a communication overhead between the server and clients. Due to this overhead, the gain is smaller than 1 for tai40b and tai50b, and is 1 for tai60b. On the other hand, gain is 1.8, 2.9, and 4.1 for tai80b, tai100b, and tai150b, respectively. This is because the communication overhead for larger problems becomes relatively smaller compared with the time required for the local search. To illustrate this, we also showed $T_{comm.}$, the total time used by the server for communication between the server and the client.

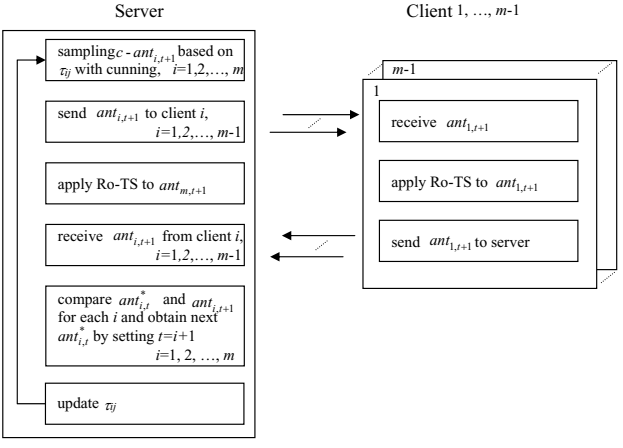

Fig. 4. Structure of load sharing p-cAS

6 Conclusions

In this paper, we applied cAS to solving the QAP and compared it agaist MMAS. The results showed cAS has promising performance. We analyzed the convergence process and the results showed that the cunning scheme is effective in maintaining diversity of pheromone density. An implementation for a simple load sharing parallel cAS (p-cAS) is also shown and a meaningful speedup of computation in the network environment was observed. However, the following study subjects remain for future work: combining cAS with other local search, such as 2OPT; study on other types of p-cAS such as the

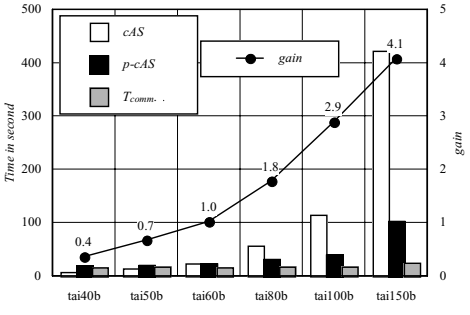

Fig. 5. Structure of load sharing p-cAS

island model; improving the server and client programs to reduce communication overhead.

Acknowledgements

This research is partially supported by the Ministry of Education, Culture, Sports, Science and Technology of Japan under Grant-in-Aid for Scientific Research number 19500199.

References

1. Tsutsui, S.: cas: Ant colony optimization with cunning ants. In: Proc. of the 9th Int. Conf. on Parallel Problem Solving from Nature (PPSN IX), pp. 162–171 (2006)
2. Tsutsui, S.: Ant colony optimization with cunning ant. Transactions of the Japanese Society for Artificial Intelligence 22(1), 29–36 (2007)
3. Stützle, T., Hoos, H.: Max-min ant system. Future Generation Computer Systems 16(9), 889–914 (2000)
4. Sahni, S., Gonzalez, T.: P-complete approximation problems. Journal of the ACM 23, 555–565 (1976)
5. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Massachusetts (2004)
6. Tsutsui, S., Liu, L.: Solving quadratic assignment problems with the cunning ant system. In: Proc. of the 2007 CEC (to appear)
7. Acan, A.: An external memory implementation in ant colony optimization. In: Dorigo, M., Birattari, M., Blum, C., Gambardella, L.M., Mondada, F., Stützle, T. (eds.) ANTS 2004. LNCS, vol. 3172, pp. 73–84. Springer, Heidelberg (2004)
8. Acan, A.: An external partial permutations memory for ant colony optimization. In: Proc. of the 5th European Conf. on Evolutionary Computation in Combinatorial Optimization, pp. 1–11 (2005)
9. Wiesemann, W., Stützle, T.: An experimental study for the the quadratic assignment problem. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds.) ANTS 2006. LNCS, vol. 4150, pp. 179–190. Springer, Heidelberg (2006)
10. QAPLIB-A Quadratic Assignment Problem Library, `http://www.opt.math.tu-graz.ac.at/qaplib/`
11. Taillard, É.D.: Robust taboo search for the quadratic assignment problem. Parallel Computing 17, 443–455 (1991)
12. Taillard, E.: Robust tabu search implementation, `http://mistic.heig-vd.ch/taillard/`
13. Cantu-Paz, E.: Efficient and Accurate Parallel Genetic Algorithm. Kluwer Academic Publishers, Boston (2000)
14. Tsutsui, S., Fujimoto, Y., Ghosh, A.: Forking gas: Gas with search space division schemes. Evolutionary Computation 5(1), 61–80 (1997)
15. Manfrin, M., Birattari, M., Stützle, T., Dorigo, M.: Parallel ant colony optimization for the traveling salesman problems. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds.) ANTS 2006. LNCS, vol. 4150, pp. 224–234. Springer, Heidelberg (2006)
16. Sun Microsystems, Inc.: Java 2 Platform, Standard Edition, v1.4.2 at API Specification, `http://java.sun.com/j2se/1.4.2/docs/api/`
17. Apache Software Foundation: Apache HTTP server project, `http://httpd.apache.org/`