Missouri Estimation of Distribution Algorithms Laboratory

Parallel Ant Colony Optimization for the Quadratic Assignment Problems with Symmetric Multi Processing

Shigeyoshi Tsutsui

MEDAL Report No. 2008006

April 2008

Abstract

Recently symmetric multi processing (SMP) has become available at a reasonable cost. In SMP, parallel run can be performed efficiently with less communication overhead among processors. In this paper, we propose several types of parallel ACO algorithms with SMP for solving the quadratic assignment problem (QAP). These models include the master-slave models and the island models. As a base ACO algorithm, we used the cunning Ant System (cAS) which showed promising performance our in previous studies. We evaluated each parallel algorithm with a condition that the run time for each parallel algorithm and the base seaquential algorithm are the same. The results suggest that using the master-slave model with increased iteration of ACO algorithms is promising in solving QAPs.

Keywords

Ant colony optimization, cunning ant system, 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/

Parallel Ant Colony Optimization for the Quadratic Assignment Problems with Symmetric Multi Processing

Shigeyoshi Tsutsui

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

Abstract. Recently symmetric multi processing (SMP) has become available at a reasonable cost. In SMP, parallel run can be performed efficiently with less communication overhead among processors. In this paper, we propose several types of parallel ACO algorithms with SMP for solving the quadratic assignment problem (QAP). These models include the master-slave models and the island models. As a base ACO algorithm, we used the cunning Ant System (cAS) which showed promising performance our in previous studies. We evaluated each parallel algorithm with a condition that the run time for each parallel algorithm and the base seaquential algorithm are the same. The results suggest that using the master-slave model with increased iteration of ACO algorithms is promising in solving QAPs.

1 Introduction

Since evolutionary computation including ant colony optimization (ACO) is a population-based search method, it is considered that the algorithm is suitable for parallelization [1] using machines such as massively parallel processors or personal computer (PC) clusters.

However, it seems that often many end-users of evolutionary algorithms use computing platforms such as generally available personal computers (PCs) or workstations (WSs) running algorithms as a single process. This is because generally available recent PCs or WSs have a relatively high performance with low cost, and using such computing platforms at hand are friendly to users–no special hardware or software configuration skills are required.

Recently, microprocessor vendors supply CPUs which have multiple cores of 2, 4, or more, and PCs or WSs which use such CPUs are available at a reasonable cost. They are normally configured with symmetric multi processing (SMP) architecture. Since the main memory is shared among processors in SMP, parallel processing can be performed efficiently with less communication overhead among processors. Thus, we can say that computing platforms running parallel algorithms using SMP to get higher performance are now ready for end users.

There are two main reasons for using parallel algorithms [2, 3]: (i) cases given a fixed time to search, to increase the quality of the solutions found within that

time; (ii) cases given a fixed solution quality, to reduce the time to find a solution not worse than that quality. In this paper, we discuss parallel ACO algorithms mainly applied to the first reason to solve the quadratic assignment problems (QAPs), with SMP type computing platforms, within the same time as is given for computing platforms with a single processor. We discuss several parallel schemes for this purpose. For this study, we use the cunning Ant System (cAS) that showed promising performance both for the traveling salesman problems (TSPs) and QAPs. In solving a QAP with an ACO algorithm, local search occupies a major portion of run time. We use this feature when we set up our parallel schemes.

The articles are structured as follows. Section 2 describes related works in brief. Section 3 gives a brief overview of the cunning Ant System (cAS) to make the paper as self-contained as possible. Then, Section 4 describes various schemes of parallel implementation of cAS in SMP for solving QAP, and the empirical analysis is given in Section 5. Finally, Section 6 concludes this paper.

2 Related Work

Many parallel ACO algorithms have been studied [2–9]. Brief summaries can be found in [3, 8]. In [2], parallel MMAS with k independent runs was studied. Time allowed for each parallel machine is set as T_{max}/k (T_{max} is time allowed for sequential runs). The experiment is performed using TSP instances of hundreds of cities. The results showed a clear advantage of parallel independent runs both in solution quality and computation time. In the research, other types of parallel models such as master/slave model are also suggested.

The most commonly used approach to parallelization is island model where multiple colonies exchange information (i.e., solution, pheromone matrix, or parameters) synchronously or asynchronously. In [5], it is repoted that the communication of the whole pheromone matrix leads to a decreased solution quality as well as worse run-time. However the exchanges of best-so-far solutions approach leads to good solution quality.

In [8], a scheme in which the whole pheromone matrix is shared with two colonies using symmetric multi processing (SMP) is studied on TSP instances of hundreds to thousands. The results showed no clear advantage of parallel ACO algorithms over sequential algorithm. In [3], parallel MMAS algorithms using message passing interface (MPI) libraries with various topologies (fully-connected, replace-worst, hypercube, ring, and independent) have been intensively studied on TSP instances of 1000 to 2000 cities, and a clear advantage of parallel algorithms is reported.

In [10], we studied a master-slave model for QAP using Java socket programming. Although with a smaller size of instances runtime decreases due to communication over the network, with a larger size of instances a clear speedup of runtime is observed.

3 An Overview of *c*AS

In previous studies we proposed a new ACO algorithm called the cunning Ant System (*c*AS), and it was applied to solving TSP [11] and QAP [12, 10]. The results showed that it could be one of the most promising ACO algorithms.

In traditional ACO algorithms, each ant generates a solution probabilistically based on the current pheromone densities $\tau_{ij}(t)$. *c*AS uses agents called cunning ants (*c-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 ant; *d-ant*) whereas the remainder of the solution is generated probabilistically from $\tau_{ij}(t)$. 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 [13–15].

Examples of *c-ant* are shown in Fig. 1 in solving TSP and QAP. For problems that contain strong nearest-neighbor dependencies such as TSP, the sampling is performed on one contiguous block of positions (remaining positions are acquired from the donor). 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 QAP, the positions to sample are selected at random with a uniform distribution over all subsets of the specified size.

In *c*AS, an elitist colony model is used In this model we maintain an archive consisting of m candidate solutions 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 *c-ant*$_{k,t+1}$(solution) is generated for each position k in the archive using the current $s_{k,t}$ i.e., solution in this position, as the donor. Then, the newly generated *c-ant*$_{k,t+1}$ is compared with its donor $s_{k,t}$ with respect to the objective function, and the best of the two survives as the next solution in this position of the archive, $s_{k,t+1}$.

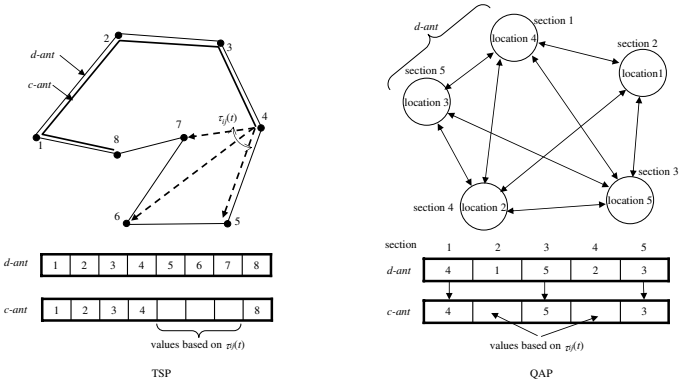

Fig. 1. *c-ant* in TSP and QAP

Pheromone density is updated with $s_{k,t+1}$ for k=1, 2, ..., m keeping all pheromone densities within the interval $[\tau_{min}, \tau_{max}]$ as in MMAS [16] and $\tau_{ij}(t+1)$ is obtained as Ant System (AS) [17].

Let us represent the number of nodes of partial solution that are constructed based on $\tau_{ij}(t)$, by l_s (i.e., l_c, the number of nodes of partial solutions from its donor, is n–l_s). Then *c*AS introduces the control parameter γ which can define

$E(l_s)$ (the average of l_s) by $E(l_s) = n \times \gamma$ and use the following probability density function $f_s(l)$ to determine l_s for each sampling.

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

cAS with *c-ant* agent and the colony model with elitism in maintaining the archive has a good balance between exploration and exploitation in a search and showed good performance both in TSP and QAP [11, 10].

4 Parallel Implementation of *c*AS in SMP for Solving QAP

4.1 Base model of the sequential *c*AS 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

$$f(\phi) = \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} a_{\phi(i)\phi(j)}, \tag{2}$$

where $A = (a_{ij})$ and $B = (b_{ij})$ are two $n \times n$ matrices and ϕ is a permutation of $\{1, 2, \ldots, n\}$. Matrix A is a distance matrix between locations i and j, and B is the flow between section i and j. 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.

In [16], when MMAS for QAP was combined with the robust taboo search (Ro-TS) by Tillard [18], parameter settings and the methods of applying Ro-TS were carefully designed so that computational time was the same as the computational times of other meta-heuristics to attain a fair comparison. These are as follows: m value of 5 and ρ value of 0.8 were used. 250 times of short Ro-TS runs of length $4n$ were applied. Since m=5, the maximum iterations are 50 (= 250/5). In a previous study in [10], we used the same setup in cAS for QAP and got promissing results.

For the base model of the sequential cAS in this work, we used the same setup with MMAS in [16] except for the value of m and the maximum number of solution constructions. Number of processors in SMP is usually an even number (power of 2), we used m value of 8, which is easy to configure for parallelization. Number of Ro-TS runs of length $4n$ is 248 (fewer than in [16] by 2) and thus the maximum iterations are 31 (= 248/8). We found the performance with $m = 8$ is slightly better than that with $m = 5$ in cAS.

4.2 Parallel Implementation of *c*AS for QAP with a SMP

In this study, we have implemented two types of parallel cAS, master-slave models and island models.

4.2.1 Master-slave models with SMP Table 1 shows computation time occupied by the local search (Ro-TS) in runs of sequential cAS with the base model described in Section 4.1. From this table, we can see that more than 99% of computational time of the base cAS is occupied by the local search. Taking into account this feature in solving QAP, we can implement a parallel model with a master-slave model as shown in Fig. 2. In this model, P threads perform local search in parallel. Here, P is number of processors of the SMP. In this research, we implemented the following two types of master-slave models.

Table 1. Computation time occupied by the local search (Ro-TS) in percentage

Problem size n	40	50	60	80	100
Local search (%)	99.75%	99.85%	99.89%	99.94%	99.96%

(1) Master-slave with longer local search: In this model, we generate P threads which can run in parallel. m solutions are generated in each iteration of cAS, we assign m/p solutions to each thread. Since we assume run time for this model is the same as the base cAS, each thread can run local search longer by P times. We identify this model as MS-LSxP.
(2) Master-Slave with longer iteration of cAS: Instead of longer run of local search of MS-LSxP, we increase maximum iteration number of cAS by P times. We identify this model as MS-ITxP.

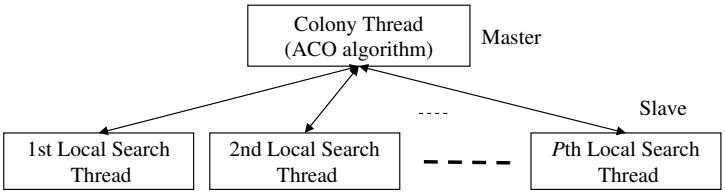

Fig. 2. Parallelization using master-slave model in SMP

4.2.2 Island models with SMP The island model is the most popular parallel processing model in evolutionary computation. Island models for genetic algorithms (GAs) are intensively studied in [1]. It is mentioned there that fully connected topologies may be the best choice when a small number of processors are available, which is the computational platform we consider in this research. cAS has an archive which maintain m solutions. This archive is similar to a population pool in GAs. In our implementation, we exchange the solutions in the archive. Although there are many topologies [1, 3], in this study we implemented the three models as described in the following. There are P colonies. Each colony is coded as a thread which can run in parallel. They exchange solutions through the control thread every I interval of iteration synchronously. In [3], asynchronous mode is also studied with MMAS for TSP in a network environment using MPI. In the SMP environment in this study, communication

overhead among processors with shared memory is small, we did not implement an asynchronous model. Fig. 3 shows our implementation.

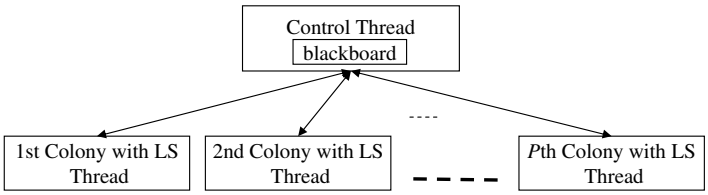

Fig. 3. Parallelization using island model in SMP

(1) Island model with *fully-connected*: In this model, the control thread collects the best solutions in the archive of each colony and chooses the best solution from them, and then distributes it to all colonies except the colony that produced that best solution. In each colony, the worst solution in each archive is replaced with the received best solution. We identify this model as IS-FCxP.
(2) Island model with *replace-worst*: This model is basically the same as (1) but the replacement is performed only in the colony which produced the worst solution among the best solutions. The worst solution in this colony is replaced by the best of all solutions of the current interval. We identify this model as IS-RWxP.
(3) Island model with *independent-runs*: P colonies simply run independently until the end of the run. When all runs are completed, the control thread collects the solutions. We identify this model by IS-INxP.

5 Experiments of Parallel *c*AS on QAP

5.1 Experimental setup

Here we evaluate parallel cAS on QAP using SMP. Setup for the base sequential cAS is described already in Section 4.1. All experiments were performed 25 times. The machine we used had two Dual Core AMD Opteron (tm) 280 (2.4GHz) processors with 2.87GB main memory (in fact it has 8GB main memory, but the OS used recognizes only up to 2.87GB), and 32-bit Windows XP. The motherboard is Supermicro® H8DCE based on the nVidia® nForce Pro 2200/2050 chipset. Thus, this machine has a total of four processors with an SMP achitecture.

In cAS, parameter value of γ (see Section 3) plays an important role. In TSP γ value of 0.4 showed better performance than other values [11]. For QAP, we tested various values of γ and found that suitable values of γ are different among differing classes QAP instances. According to [19, 16], QAP instances in QAPLIB [20] can be divided into 4 classes; (i) randomly generated instances, (ii) grid-based distance matrix instances, (iii) real-life instances, and (iv) real-lifelike instances. In Fig. 4, we show examples of changes of solution quality by *Error* (%), i.e., average of 25 independent runs of (function value – known best)/known

best, of tai80a, sko72, ste36a, and tai80b as a representative of QAP classes (i), (ii), (iii), and (iv), respectively. In the figure, each central square indicates *Error* and upper and lower points indicate range of ± *Std* (standard deviation) from the value of *Error*.

From this figure, we can see that instances classified into (i) and (ii) have better results with smaller values of γ, on the contrary, instances classified into (iii) and (iv) have better results with larger values of γ. Thus, in experiments in Section 5 we used γ value of 0.4 for QAP instances classified into (i) and (ii), and 0.8 for (iii) and (iv), respectively. We selected four instances for each class from QAPLIB.

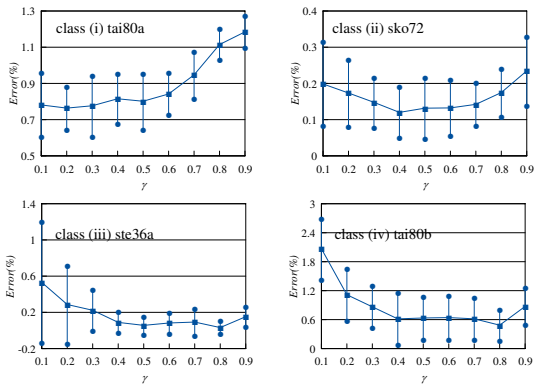

Fig. 4. Change of *Error* of sequential *c*AS for various γ values

*c*AS was written in Java and the thread programming uses Java Thread class. For run, Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_13-b06) was used. We used the Ro-TS code which is available at [18], though the code, which is originally written in C, was rewritten in Java.

To attain the same run time between parallel *c*AS on the SMP machine and the base *c*AS on the same machine, we experimented in the following manner.

(1) The base model: The base model uses the setup described in Section 4.1, and was run as a single process. Although the SMP has four processors, we run only one *c*AS on the machine to see the exact run time for single runs. This model becomes the base to see the solution quality and real run time.

(2) Master-slave model: We ran MS-LSxP and MS-ITxP with P=4. Thus, these models are indicated as MS-Lx4 and MS-Ix4, respectively.

(3) Island model: We ran IS-INxP, IS-FCxP abd IS-RWxP with P=4 with solutions exchange interval I=2. These models are indicated as IS-INx4, IS-FCx4 and IS-RWx4, respectively.

All of these models should have almost the same run time (see Section 4.2) if the machine has a well-designed architecture.

5.2 Analysis of the results

Results are summarized in Table 2 for classes (i) and (ii), and Table 3 for classes (iii) and (iv), respectively. In each table, the notations are as follows:

Error(%) indicates the solution quality calculated by the average of (function value – known best)/known best over 25 independent runs.

R-time (ratio) indicates average run time of algorithm of each model in seconds. Values in brackets are ratios of the run time of the base model.
#OPT indicates the number of runs in which algorithms succeeded in finding the known best solution.
O-time (ratio) indicates the average time to find the known bast solution in those runs where it did find it, where the values in brackets are ratios of the *O-time* of the base model.
B-time (ratio) indicates the average time at which the best functional values of each run was found, where the values in brackets indicate ratios of the *B-time* of IS-INx4.

Table 2. Results of parallel *c*AS on QAP on classes (i) and (ii)

QAP	Measure	Base Model	Master-Slave Model		Island Model		
			MS-LSx4	MS-ITx4	IS-INx4	IS-FCx4	IS-RWx4
class (i) with γ=0.4							
tai40a	*Error (%)*	0.789	**0.495**	0.630	0.587	0.566	0.569
	Std	0.217	0.183	0.151	0.164	0.157	0.146
	R-time (ratio)	6.81 (1)	(1.00)	(1.04)	(1.00)	(1.03)	(1.01)
	#OPT	0	1	0	0	0	0
	O-time (ratio)	-	0.92	-	-	-	-
	B-time (ratio)	-	**(0.85)**	(0.93)	3.83 (1)	(0.85)	(1.16)
tai50a	*Error (%)*	1.064	**0.738**	0.830	0.832	0.804	0.773
	Std	0.166	0.222	0.223	0.184	0.179	0.192
	R-time (ratio)	13.44 (1)	(0.99)	(1.03)	(1.00)	(1.01)	(1.01)
	#OPT	0	1	0	0	0	0
	O-time (ratio)	-	11.56	-	-	-	-
	B-time (ratio)	-	**(0.82)**	(0.82)	8.92 (1)	(0.92)	(1.04)
tai60a	*Error (%)*	1.101	**0.881**	0.930	0.913	0.899	0.886
	Std	0.124	0.141	0.112	0.174	0.196	0.173
	R-time (ratio)	23.34 (1)	(0.99)	(1.02)	(1.00)	(1.01)	(1.01)
	#OPT	0	0	0	0	0	0
	O-time (ratio)	-	-	-	-	-	-
	B-time (ratio)	-	**(0.92)**	(1.00)	16.65 (1)	(0.69)	(1.00)
tai80a	*Error (%)*	0.815	0.592	**0.547**	0.672	0.581	0.585
	Std	0.137	0.147	0.122	0.093	0.132	0.126
	R-time (ratio)	56.73 (1)	(0.99)	(1.02)	(1.01)	(1.00)	(1.01)
	#OPT	0	0	0	0	0	0
	O-time (ratio)	-	-	-	-	-	-
	B-time (ratio)	-	(0.98)	**(0.84)**	36.81 (1)	(1.19)	(1.23)
class (ii) with γ=0.4							
sko49	*Error (%)*	0.057	0.034	**0.024**	0.035	0.030	0.031
	Std	0.040	0.030	0.029	0.030	0.030	0.029
	R-time (ratio)	12.57 (1)	(0.99)	(1.02)	(1.00)	(1.00)	(1.02)
	#OPT	6	10	14	10	12	11
	O-time (ratio)	6.89 (1)	**(0.80)**	(0.91)	(1.19)	(1.01)	(1.04)
	B-time (ratio)	-	**(0.82)**	(0.88)	7.37 (1)	(1.01)	(1.08)
sko64	*Error (%)*	0.072	**0.005**	0.011	0.009	0.005	0.006
	Std	0.072	0.009	0.015	0.014	0.012	0.009
	R-time (ratio)	28.47 (1)	(0.98)	(1.01)	(1.00)	(1.00)	(1.00)
	#OPT	1	10	8	13	17	16
	O-time (ratio)	20.20 (1)	(0.89)	**(0.61)**	(0.77)	(0.71)	(0.78)
	B-time (ratio)	-	(1.03)	**(0.78)**	15.47 (1)	(0.96)	(1.00)
sko81	*Error (%)*	0.129	**0.044**	0.048	0.076	0.049	0.059
	Std	0.045	0.024	0.032	0.022	0.028	0.024
	R-time (ratio)	58.91 (1)	(0.99)	(1.02)	(1.01)	(1.01)	(1.01)
	#OPT	0	0	1	0	0	0
	O-time (ratio)	-	-	33.92	-	-	-
	B-time (ratio)	-	(0.96)	(0.97)	43.53 (1)	**(0.93)**	(1.09)
sko90	*Error (%)*	0.155	0.074	0.095	0.085	**0.050**	0.063
	Std	0.067	0.047	0.063	0.043	0.039	0.036
	R-time (ratio)	81.04 (1)	(1.00)	(1.03)	(1.01)	(1.01)	(1.01)
	#OPT	0	0	0	1	2	1
	O-time (ratio)	-	-	-	57.860	39.36	13.25
	B-time (ratio)	-	(0.92)	**(0.91)**	59.77 (1)	(1.05)	(1.12)

Before analyzing the results, let us see values of the *R-time (ratio)* of each experiment. We can see that each run time of the parallel model attained almost the same value as the run time of its base model, i.e., the value of *ratio* is very near to 1. Thus, we can say that the SMP platform used in this research

executed the algorithms in parallel without significant performance degradation. Overhead caused by the synchronous solutions exchanges with interval 2 in the island models was also negligible.

All results with the parallel models show increases of solution quality as compared to the quality of their corresponding base models. In the following, we analyze these results for each problem class.

From the results of class (i) instances in Table 2, we can see the values of *Error* of master-slave models are slightly better than those of the island models. But in QAP instances of this class (randomly generated instances), there are not such significant differences among the five parallel models, though the results of the island model with independent runs (IS-INx4) show the worst *Error* values among island models for all instances in this class. However, if we look at the *B-time*, the master-slave models show smaller *B-time*. This means the master-slave models find their best solutions faster in each run than island models.

From the results of class (ii) instances (grid-based distance matrix instances) in the same table, we can see the values of *Error* of master-slave models are slightly better than those of the island models except for sko90. Again IS-INx4 shows the worst *Error* values among island models for all instances in this class. Comparisons with *O-time* are possible on sko49 and sko64. On these instances, the master-slave models have smaller values of *O-time* than the island models, showing faster finding of the known best solutions.

Results of class (iii) instances (real-life instances) in Table 3 showed a different feature as compared to the results of classes (i) and (ii) in Table 2. In class (iii), values of $\#OPT$ for all parallel models attained 25 except for on ste36a. So, we cannot see differences in *Error* values among parallel algorithms. Instead let us turn our attention to *O-time (ratio)*. For example in ste36b, $\#OPT$ of all models including the base model is is 25. On the other hand, looking at the *O-time* values of the parallel model, the ratios of *O-time* against the base model are 0.50 (MS-LSx4), 0.31 (MS-ITx4), 0.63 (IS-INx4), 0.57 (IS-FCx4), and 0.63 (IS-RWx4), respectively. Thus, we can see that the master-slave models perform faster searches than the island models on this instance. Other instances in this class except for kra30a showed similar results. On kra30a, island models find known best solution faster than the master-slave models.

Results of class (iv) instances (real-lifelike instances) in Table 3 showed similar results to that of class (iii), though values of $\#OPT$ in this class are smaller than those of (iii) instances except for tai40b. Results for tai40b are almost the same as instances of class (iii). Although on tai60b, the values of *Error* of all iland models are smaller than the values of *Error* of master-slave models, the values of *O-time* of island models are much larger tha those of master-slave models. On the all instances in this class, MS-ITx4 has the smallest *O-time* values among parallel models. Thus, the advantage of MS-ITx4 can be observed in this class. We can also observe that the *B-time* of the MS-ITx4 is the smallest among all parallel models.

As a summary of analysis in this section, we can say master slave models, especially MS-ITx4, showed the clear advantage for instances in classes (iii) and

(iv) on the measures of *Error*, *O-time*, and *B-time*. Although we could not observe which parallel models are the best for instances in classes (i) and (ii) by looking at *Error* values, *B-time* analysis suggests that using master-slave models is a better choice than island models.

Table 3. Results of parallel *c*AS on QAP on classes (iii) and (iv)

QAP	Measure	Base Model	Master-Slave Model		Island Model		
			MS-LSx4	MS-ITx4	IS-INx4	IS-FCx4	IS-RWx4
class (iii) with γ=0.8							
kra30a	*Error (%)*	0.054	**0**	**0**	**0**	**0**	**0**
	Std	0.268	0	0	0	0	0
	R-time (ratio)	2.84 (1)	(1.00)	(1.05)	(1.03)	(1.00)	(0.99)
	#OPT	24	25	25	25	25	25
	O-time (ratio)	0.88 (1)	(0.81)	(0.65)	(0.52)	**(0.49)**	(0.52)
	B-time (ratio)	-	(1.55)	(1.25)	0.46 (1)	**(0.94)**	(0.99)
kra30b	*Error (%)*	0.009	**0**	**0**	**0**	**0**	**0**
	Std	0.025	0	0	0	0	0
	R-time (ratio)	2.84 (1)	(1.00)	(1.05)	(1.01)	(0.99)	(1.01)
	#OPT	22	25	25	25	25	25
	O-time (ratio)	1.46 (1)	(0.51)	**(0.35)**	(0.47)	(0.58)	(0.47)
	B-time (ratio)	-	(1.09)	**(0.74)**	0.69 (1)	(1.24)	(1.00)
ste36a	*Error (%)*	0.032	**0.004**	0.010	0.025	0.023	0.004
	Std	0.072	0.021	0.050	0.028	0.021	0.028
	R-time (ratio)	4.94 (1)	(1.00)	(1.04)	(1.00)	(1.01)	(1.00)
	#OPT	20	24	24	23	24	23
	O-time (ratio)	2.96 (1)	**(0.59)**	(0.64)	(0.79)	(0.86)	(0.79)
	B-time (ratio)	-	**(0.83)**	(0.98)	2.28 (1)	(1.09)	(0.97)
ste36b	*Error (%)*	**0**	**0**	**0**	**0**	**0**	**0**
	Std	0	0	0	0	0	0
	R-time (ratio)	4.94 (1)	(1.00)	(1.04)	(1.01)	(1.01)	(1.00)
	#OPT	25	25	25	25	25	25
	O-time (ratio)	1.42 (1)	(0.50)	**(0.31)**	(0.63)	(0.57)	(0.63)
	B-time (ratio)	-	(0.78)	**(0.48)**	0.89 (1)	(0.90)	(0.98)
class (iv) with γ=0.8							
tai40b	*Error (%)*	**0**	**0**	**0**	**0**	**0**	**0**
	Std	0	0	0	0	0	0
	R-time (ratio)	6.76 (1)	(1.01)	(1.04)	(1.01)	(1.01)	(1.01)
	#OPT	25	25	25	25	25	25
	O-time (ratio)	3.16 (1)	(0.90)	**(0.26)**	(0.50)	(0.40)	(0.44)
	B-time (ratio)	-	(1.79)	**(0.53)**	1.58 (1)	(0.80)	(0.88)
tai60b	*Error (%)*	0.071	0.069	0.046	0.009	0.002	**0.001**
	Std	0.138	0.142	0.128	0.013	0.011	0.004
	R-time (ratio)	23.17 (1)	(0.99)	(1.02)	(1.01)	(1.00)	(1.00)
	#OPT	4	9	22	9	23	21
	O-time (ratio)	22.61 (1)	(0.86)	**(0.35)**	(0.93)	(0.76)	(0.86)
	B-time (ratio)	-	(0.93)	**(0.41)**	20.79 (1)	(0.84)	(0.96)
tai80b	*Error (%)*	0.484	0.671	**0.208**	0.226	0.262	0.196
	Std	0.321	0.455	0.264	0.162	0.328	0.238
	R-time (ratio)	56.58 (1)	(0.99)	(1.03)	(1.00)	(1.01)	(1.01)
	#OPT	0	0	9	0	2	0
	O-time (ratio)	-	-	**34.35 (1)**	-	(1.67)	-
	B-time (ratio)	-	(1.11)	**(0.81)**	50.68 (1)	(1.07)	(1.08)
tai100b	*Error (%)*	0.154	0.138	**0.055**	0.112	0.075	0.102
	Std	0.14	0.08	0.08	0.086	0.060	0.071
	R-time (ratio)	114.89 (1)	(0.98)	(1.00)	(1.01)	(1.01)	(1.01)
	#OPT	0	0	13	0	3	0
	O-time (ratio)	-	-	**50.77 (1)**	-	(1.99)	-
	B-time (ratio)	-	(0.96)	**(0.61)**	106.35 (1)	(1.00)	(1.03)

5.3 Comparison of SMP and network model run time

Results in Section 5.2 suggest that using the master slave models with SMP to solve QAP would be a good choice. Although this paper pays main attention to evaluating the parallel ACO algorithms using SMP, we have other options for parallelization of ACO algorithms, such as the cluster system connected by a network as studied in [3]. In this section, we compare the master-slave models that use SMP (MS-SMP) and simple master-slave models which use a network (MS-NET). The MS-NET use two machines, the same machines as were used

in the experiments in Section 5.2, each having four processors. One machine performs ACO algorithms as the master and the other machine performs the local search task. The two machines are connected via a 1000BASE-T switching hub.

Table 4 shows the run time comparisons with the MS-SMP and the MS-NET using class (iv) instances. The values of run time are averaged over 25 runs and are represented by the ratios of the run time of the base model. From this table, we can see a significant increase of run time with the MS-NET. Especially, the increases of run time ratios are significant for MS-ITx4 with smaller instances. This is because the communication overhead occupies a larger portion of total run time in these situations.

Table 4. Comparison of run time between MS-SMP and MS-NET

QAP	Base model	SMP		Network	
		MS-LSx4	MS-ITx4	MS-LSx4	MS-ITx4
tai40b	6.76 (1)	(1.01)	(1.04)	(2.50)	(8.72)
tai60b	23.17 (1)	(0.99)	(1.02)	(1.50)	(3.08)
tai80b	56.58 (1)	(0.99)	(1.02)	(1.13)	(1.68)
tai100b	113.09 (1)	(0.99)	(1.03)	(1.08)	(1.29)

6 Conclusions

In SMP, parallel run can be performed efficiently with less communication overhead among processors. In this paper, we proposed a total of five parallel ACO algorithms with SMP for solving QAP. We evaluated each parallel algorithm under conditions in which the run time for each parallel algorithm and the base algorithm are about the same.

The results showed all parallel models studied here improved solution quality of the base algorithm. Although detail improvement features were different depending on problem classes of QAPs, our results suggested that using the master-slave models, especially MS-ITxP for classes (iii) and (iv), is promising in solution quality and search speed for solving QAPs where computation for local search occupies a large part of the total computation time.

To study other parallel models, such as a combination of the master-slave models and island models remains for future work. To apply these models to problems in other domains also remains for future work.

Acknowledgements

The author would like to acknowledge to Ms. Lichi Liu and Prof. Gordon Wilson for their useful discussion on this research. 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. Cantu-Paz, E.: Efficient and Accurate Parallel Genetic Algorithm. Kluwer Academic Publishers, Boston (2000)

2. Stützle, T.: Parallelization strategies for ant colony optimization. 5th International Conf. on Parallel Problem Solving for Nature (PPSN-V) (1998) 722–731
3. 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
4. Randall, M., Lewis, A.: A parallel implementation of ant colony optimization. Journal of Parallel and Distributed Computing **62**(9) (2002) 1421–1432
5. Benkner, S., Doerner, K., Hartl, R., Kiechle, G., Lucka, M.: Communication strategies for parallel cooperative ant colony optimization on clusters and grids. Complimentary Proc. of PARA'04 Workshop on State-of-the-art in Scientific Com-puting (2005) 3–12
6. Bullnheimer, B., Kotsis, G., Straussb, C.: Parallelization strategies for the ant system. High Performance Algorithms and Software in Nonlinear Optimization (1998) 87–100
7. Middendorf, M., Reischle, F., Schmeck, H.: Multi colony ant alorithms. Journal of Heuristics **8**(3) (2002) 3005–320
8. Lv, Q., Xia, X., Qian, P.: A parallel aco approach based on one pheromone matrix. Proc. of the 5th Int. Workshop on Ant Colony Optimization and Swarm Intelligence (ANT 2006) (2006) 332–339
9. Talbi, E., Roux, O., Fonlupt, C., Robillard, D.: Parallel ant colonies for the quadratic assignment problem. Future Generation Computer System **17** (2001) 441–449
10. Tsutsui, S.: Cunning ant system for quadratic assignment problem with local search and parallelization. Proc. of the Second International Conference on Pattern Recognition and Machine Intelligence, Springer LNCS 4815 (2007) 269–278
11. Tsutsui, S.: *c*AS: Ant colony optimization with cunning ants. Proc. of the 9th Int. Conf. on Parallel Problem Solving from Nature (PPSN IX) (2006) 162–171
12. Tsutsui, S., Liu, L.: Solving quadratic assignment problems with the cunning ant system. Proc. of the 2007 IEEE CEC (2007)
13. 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
14. 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
15. 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
16. Stützle, T., Hoos, H.: Max-min ant system. Future Generation Computer Systems **16**(9) (2000) 889–914
17. 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
18. Taillard, É.D..: (Robust tabu search implementation) http://mistic.heig-vd.ch/taillard/.
19. Taillard, É.D.: Robust taboo search for the quadratic assignment problem. Parallel Computing **17** (1991) 443–455
20. QAPLIB-A Quadratic Assignment Problem Library: (http://www.opt.math.tu-graz.ac.at/qaplib/)