Island Model Parallel Hybrid-GA for Large Scale Combinatorial Optimization

Tang Jing[1], Meng Hiot Lim[1], and Yew Soon Ong[2]
[1]School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798
{pg04159923, emhlim}@ntu.edu.sg
[2]School of Computer Engineering, Nanyang Technological University, Singapore 639798
asysong@ntu.edu.sg

Abstract

In this article, we describe a parallel hybrid-GA (PHGA) for combinatorial optimization using an island model running in a networked computing environment. In particular, we focus on quadratic assignment problem (QAP), among the hardest combinatorial optimization problems known. The solutions of these problems require both improvement in mathematical programming algorithms and the utilization of powerful computational platforms. QAP benchmarks of high complexity for n ranging from 60 to 256 are simulated. Results show that a two-island PHGA employing a simplistic elite migration between islands outperforms the serial hybrid-GA (SHGA) significantly.

1. Introduction

The quadratic assignment problem (QAP) is a class of combinatorial optimization problems with many interesting practical applications. It was formulated by Koopmans and Beckmann [5] for location planning of economic activities. To formulate a QAP mathematically, consider n facilities to be assigned to n locations with minimum cost. The QAP can be described by two $n \times n$ matrices $A = [a_{ij}]$ and $B = [b_{ij}]$. The goal is to find a permutation π of the set $M=\{1,2,3,\ldots,n\}$,which minimizes the objective function $C(\pi)$ as in Eq.(1).

$$C(\pi) = \sum_{l=1}^{n}\sum_{t=1}^{n} a_{lt} b_{\pi(l)\pi(t)}$$

In the above equation, matrix A can be interpreted as a distance matrix, i.e. a_{ij} denotes the distance between location i and location j, and B is referred to as the flow matrix, i.e. b_{ij} represents the flow of materials from facility i to facility j. We represent an assignment by the vector π. $\pi(i)$ is the location to which facility i is assigned.

There are many applications that can be formulated as QAP, such as scheduling parallel production lines, machine scheduling, facility layout problem and so on. Its availability has elicited much research focused on the optimization algorithm [1, 2, 3, 7, 8, 9, 11].

The main aim of this paper is to propose the parallel hybrid-GA model inspired by the current interest in grid computing technology, and to utilize the new PHGA model to solve the QAP. Embedded in a distributed environment, PHGA is expected to treat the large QAPs more efficiently and effectively. Experience will help us to evaluate the performance of our new model and to make some improvements on the new model. The simulation results presented in this paper will show that our new model is more efficient and effective than that in our previous work [1,2,3].

This paper is organized as follows. Section 2 provides an introduction to PGA, giving a brief overview of the multiple-population model in particular. Our new model, a two-island PHGA model is proposed in Section 3. Section 4 focuses on a grid-enabled solver for our algorithm. Section 5 summarizes our experimental results comparing serial GA with our new model on the QAP. Finally, Section 6 concludes the paper with some directions for future work.

2. Parallel Genetic Algorithm

GA is an iterative procedure which borrows from the idea of natural selection and 'survival of the fittest' of natural evolution. By simulating natural evolution, a GA can be used to solve complex problems. Furthermore, by emulating biological selection and reproduction techniques, a GA can effectively search the problem domain in a general, problem-independent manner.

It can be seen that GA besides being easily handled, is extensible and has been shown to produce satisfactory performance in many cases. However, for complex combinatorial optimization, the computational cost incurred is still on the high side. It can be demanding in terms of computation load and memory. To address this issue, the parallel GA is proposed by some researchers.

Parallel GA is an extension of basic GA. The most important advantage of parallel GAs is that in many cases, the multi-population GAs provide better performance than single population-based algorithms, even when the parallel GA is simulated sequentially. The reason is that multiple

populations permit speciation, a process by which different populations evolve in different directions. For this reason parallel GAs are not only an extension of the traditional GA sequential model, but they represent a new class of algorithms in that they search the space of solutions differently.

Parallel GAs have been developed to speed up this progress as well as to obtain higher quality solutions when dealing with complex problems. Much research has been on PGA. There are three main types of parallel GAs, (1) global single-population master-slave GAs, (2) single-population fine-grained GAs, and (3) multiple-population coarse-grained GAs. Further detail on this can be found in [12]. In particular, we utilize the multi-population coarse-grained GAs in our algorithm.

Multiple-population (or multiple-deme) GAs are very sophisticated, as they consist of several subpopulations which exchange individuals occasionally. This exchange of individuals is called migration and it is controlled by several parameters. Multiple-deme parallel GAs are known with different names. Sometimes they are known as "distributed" GAs, because they are usually implemented in distributed-memory MIMD computers. Since the computation to communication ratio is usually high, they are occasionally called coarse-grained GAs. Finally, multiple-deme parallel GAs resemble the "island model" in population genetics which considers relatively isolated demes, so the parallel GAs are also known as "island" parallel GAs.

In multiple-deme parallel GAs, the arrival of individuals from other populations can trigger evolutionary changes. It is also noticed that there was relatively little change between migrations, but new solutions were found shortly after individuals were exchanged. Multiple-deme GAs, although popular, is also a class of parallel GAs is probably the most difficult to understand, because the effects of migration are not fully understood. Migration is controlled by several parameters [14]:

- Migration rate determines how many individuals migrate from a population.
- Migration frequency (migration interval) determines how often migrations occur.
- Migration topology determines the destination of the migrants.
- Migration policy determines which individuals migrate and which are replaced at the receiving deme.

The island model strategy eliminates the global synchronization that is required in the panmictic approach. However, in the majority of multi-deme parallel GAs, some synchronization is usually required for migration which means that it occurs at predetermined constant intervals.

3. Island Model for QAP

In our earlier work [1,2,3], serial GA has been proposed to solve the QAP. The algorithm is a hybrid-GA with stochastic selection local search which is a form of *k*-gene exchange. All the reported solutions are quite satisfactory. However, the computational time of the algorithm is much higher, especially for larger size problems, $n \geq 100$.

In our work, we utilized the island model to implement the parallel GA. Figure 1 illustrates the model we used.

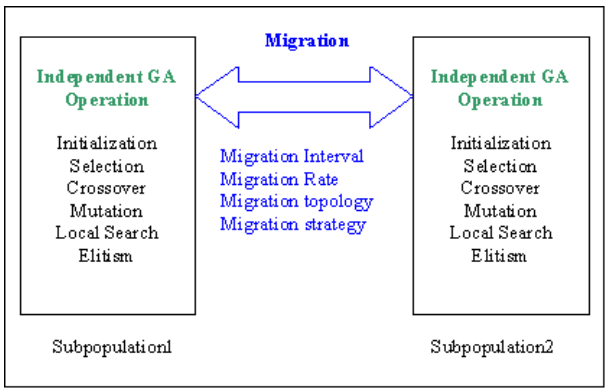

Figure 1 Two-island PHGA Model

As described in Figure 1, we carry out independent GA in two subpopulations, hence the two-island PHGA model. We implement migration between the two subpopulations according to the corresponding migration parameters.

Our parallel GA maintained a total population size of 200, as used in the serial GA. Using migration operation, the PHGA provided exceptionally better results than running genetic algorithm in each subpopulation independently, an observation made by the authors in preliminary work. It is concluded in [13] that the total amount of work in the parallel GA is less than the work that a serial GA requires to reach the same solution. It is believed that the migration policies increase the selection pressure and cause the parallel GA to converge faster than would be expected from dividing a large population into smaller demes. One additional note is that convergence rate depends on the GA parameters and migration interval. When migration occurs too frequently, the subpopulations tend to converge very quickly, and rarely do they produce satisfactory results. When migration is too infrequent, it may not be enough to prevent each small subpopulation from prematurely converging. In addition, in the early stage of the PHGA, the number of correct building blocks in the migrants may be too small to influence the search in the right direction. Hence, the migration should occur precisely when the genetic algorithm has reached convergence on the regions [15]. Considering all the above, we design our algorithm to carry out the migration every 10 generations at the early stage, and every 5 generations at the later stage. Each time we exchange one individual, and choose the best one in each subpopulation to replace the worst one in another subpopulation. This migration strategy we have adopted in this work shows encouraging result in Section 5.

4. The Grid-enabled Solver

The performance of parallel GAs is desirable. However, traditional parallel processing (super) computers have inherent limitations arising from their expense and lack of

availability. An alternative platform for massively distributed computation is based on the notion of grid computing [6]. A computational grid consists of a potentially large number of geographically dispersed CPUs linked by a communication medium such as the Internet. The advantage of such a platform compared to a traditional multiprocessor machine is that a large number of CPUs may be assembled very inexpensively. In our work, the PHGA is implemented on NetSolve, a tool based on the grid computing technology.

NetSolve [4] is an example of a grid computing system. It is a system designed to allow the use of network resources of numerical computing by providing convenient access to problem-solving techniques (software) in addition to managing computational resources. The basic objective is to provide a uniform, portable and efficient way to access computational resources over a network. It provides remote access to computational resources, both hardware and software. NetSolve serves as a grid middleware providing ease of use of general purpose Scientific Computing Environments (e.g. Matlab and Mathematica), which are essential tools for the majority of working scientists and engineers.

The NetSolve system comprises of a set of loosely connected machines. By loosely connected, it means that these machines are on the same local, wide or global area network, and may be administrated by different institutions and organizations. Moreover, the NetSolve system is able to support these interactions in heterogeneous environment, i.e. machines of different architectures, operating systems and internal data representations can participate in the system at the same time. NetSolve is a powerful tool to support the implementation of PHGA. Its favorable mechanism and flexible framework will facilitate the implementation of the algorithm efficiently and conveniently.

5. Computational results on large problems

Our previous work [1,2,3] shows that the serial hybrid GA provides good performances for solving QAP. However, it takes a long time to converge on QAP problems of high complexity. The two-island PHGA model, implemented using grid technology in a distributed framework, represents the parallel hybrid GA. In this section, our attempt to improve the impact of the two-island PHGA model implemented using Netsolve is studied by making comparison to the serial hybrid GA on several QAP benchmarks of high complexity.

5.1 Simulation setup

In this paper, we compare the performance of the serial hybrid GA with two-island PHGA both in terms of computation time and solution quality. The two models, serial GA and parallel GA, are used to solve several QAP benchmarks. For convenience, serial hybrid-GA is denoted as SHGA, while PHGA indicates the parallel hybrid-GA.

The programs were coded in C programming language and the simulations were carried out on a cluster of Pentium IV 1.9 GHz workstations. Each machine is equipped with 256Mb of RAM, and running the Linux Redhat 7.0 operating system. For each benchmark problem, we conduct 10 simulation runs to obtain the statistical result. For the serial GA, the parameters are set as follows: Population size = 200; Maximum number of generations = 500; Fitness scaling factor S_f = 3; Crossover probability P_c = 0.8; Mutation probability P_m = 0.05; Zerofit threshold constant K_z = 5 and Elitists size = 6.

For the parallel GA, the following parameters were used: Total population size = 200; Subpopulation size = 100; Maximum number of generations = 500; Fitness scaling factor S_f = 3; Crossover probability P_c = 0.8; Mutation probability P_m = 0.05; Zerofit threshold constant K_z = 5; Elitists size in each subpopulation = 3.

The migration policy is based on a two-stage progressive increase in migration interval. For the initial 100th generation, we carried out migration every 10 generations. Subsequently, migration frequency is increased to every 5 generations. The migration rate is one chromosome per migration phase. At this stage, the two-island PHGA is parallelized across two computers, so migration involves exchanges between the two machines. The migration strategy used here is the simple elitist strategy, where the best individual in one subpopulation replaces the worst in the other.

In both SHGA and PHGA, the algorithms are terminated if any of the following criteria are met: solution stalls for more than 70 successive generations or maximum number of generations has been reached. To evaluate the two algorithms, several criteria based on solution quality and computation time have been introduced. The abbreviations used in Table 1-4 are defined as follows.

CPU time is the average computation time in seconds upon termination of the algorithm. It is used to measure the speed of the algorithm in real time.

Generation is the average number of generations elapsed before the occurrence of the best solution while *TG* is the average number of generations elapsed when the algorithm terminates. The two parameters measure the convergence speed of the algorithm in terms of the number of iterations rather than the real time.

Average refers to the average objective value of the solutions obtained for all the GA runs and *Average gap* is the difference between the *Average* and the best known value of the objective function. These parameters serve as the criteria for measuring the solution quality of the algorithm. In addition, we define *Best* as best solution obtained among all the GA runs and *Gap* as the difference between the best-found value and the best-known value of a benchmark problem. *Success rate* refers to the number of times the algorithm finds the best-known solution out of all the GA runs. To compare the PHGA with the SHGA in aspects of both solution quality and the computational time, we define four other parameters, R_1, R_2, R_3 and R_4. R_1 indicate the percentage improvement in solution quality of the PHGA to that of the SHGA. A positive value of R_1 indicates an improvement in terms of solution quality while the negative value represents deterioration. R_2 refers to the percentage reduction in the *Average gap of* the PHGA to that of the SHGA. R_3 is the percentage improvement in CPU time for the PHGA to that of the

SHGA, while R_4 is the percentage improvement in CPU time for each generation of the PHGA with respect to the SHGA.

5.2 Simulation results and analysis

Tables 1 to 3 summarize the simulation results on *sko*, *lipa* and *tai* type of benchmark problems, respectively. They are classes of synthetic problems randomly generated to study the robustness of algorithms for solving QAPs. The size of all these problems is relatively large and the simulation results are presented in the order of problem size. We compare the performance both in

		CPU time	Generation	TG	Average	Average gap	Best	Gap	Success rate	R1	R2	R3	R4
sko64	PHGA	90.00	157.20	227.20	48613.60	0.24%	48508	0.02%	0.00%	0.146%	38.05%	-67.91%	-120.01%
48498	SHGA	53.60	227.40	297.70	48684.60	0.39%	48550	0.11%	0.00%				
sko72	PHGA	94.30	154.40	224.40	66469.40	0.32%	66306	0.08%	0.00%	0.134%	29.48%	-38.80%	-65.77%
66256	SHGA	67.94	198.00	268.00	66558.60	0.46%	66330	0.11%	0.00%				
sko81	PHGA	107.20	154.50	224.50	91209.60	0.23%	91072	0.08%	0.00%	0.172%	42.56%	35.54%	18.63%
90998	SHGA	166.30	213.40	283.40	91366.40	0.40%	91118	0.13%	0.00%				
sko90	PHGA	164.20	216.10	286.10	115928.20	0.34%	115640	0.09%	0.00%	0.181%	34.76%	37.45%	47.31%
115534	SHGA	262.50	171.00	241.00	116138.20	0.52%	115884	0.30%	0.00%				
sko100a	PHGA	194.00	203.40	273.40	152322.80	0.21%	152122	0.08%	0.00%	0.234%	52.74%	52.10%	55.71%
152002	SHGA	405.03	183.80	252.80	152680.80	0.45%	152104	0.07%	0.00%				
sko100b	PHGA	183.60	171.80	241.80	154215.00	0.21%	153924	0.02%	0.00%	0.224%	51.58%	43.29%	40.19%
153890	SHGA	323.73	185.00	255.00	154561.20	0.44%	154122	0.15%	0.00%				
sko100c	PHGA	184.40	205.80	275.80	148140.40	0.18%	148050	0.13%	0.00%	0.095%	33.65%	69.77%	63.29%
147862	SHGA	610.09	265.00	335.00	148281.60	0.28%	148054	0.13%	0.00%				
sko100d	PHGA	232.10	259.90	327.40	150036.80	0.31%	149732	0.10%	0.00%	0.228%	42.72%	48.39%	60.61%
149576	SHGA	449.70	179.90	249.90	150380.40	0.54%	150142	0.38%	0.00%				
sko100e	PHGA	235.50	252.90	322.90	149642.20	0.33%	149198	0.03%	0.00%	0.226%	40.80%	28.99%	49.03%
149150	SHGA	331.66	161.80	231.80	149981.40	0.56%	149400	0.17%	0.00%				
sko100f	PHGA	206.50	214.80	284.80	149496.60	0.31%	149228	0.13%	0.00%	0.270%	46.74%	40.69%	44.37%
149036	SHGA	348.16	197.10	267.10	149900.80	0.58%	149570	0.36%	0.00%				

Table 1 Testing results of *sko* type benchmarks

terms of the solution quality and the improvement in execution time.

Table 1 summarizes the simulation results of several *sko* type benchmark problems. In the table, we can observe that the parallel GA outperforms the serial GA significantly in both solution quality and computation time, especially on large *sko* problems, i.e. n>81. However, on smaller problems, *sko64* and *sko72*, parallel GA consumes more time than the serial GA. Compared to larger problems, the communication overhead of smaller problems incurs a larger proportion of the total CPU time, as compared to the computational time. Hence the communication overhead dominates the computational cost. This results in the parallel GA being computationally less efficient on smaller problems. Nevertheless, on larger-size problems, the computation time takes up a large proportion of the total CPU time while the communication overhead for migration is in comparison a relatively small proportion.

As for the *lipa* type QAP benchmarks, Table 2 shows that both SHGA and PHGA perform well on the *lipa-a* QAP benchmark problems. Nevertheless, the parallel GA has some improvement in the *Average gap* as compared to the serial GA. On the *lipa-b* QAP benchmark problems, the results indicate that the success rate of PHGA is much higher than SHGA. In addition, for the modest or smaller size problems shown in Table 2, the PHGA is computationally less efficient as we would expect. This is because the communication overhead takes up a large proportion of the total time on modest or small size problems. Thus the parallel GA is not very appropriate for *lipa* type QAP benchmarks of smaller size.

Table 3 shows the simulation results of several modest or larger-sized *tai* type QAP benchmark problems. From Table 3, it is apparent that for all the benchmarks, the solution quality using PHGA is improved significantly. But in terms of computation time, on smaller size problems, the parallel GA proves once again to be computationally inefficient. While on larger size problems, the parallel GA produces better solutions than the serial GA at a short CPU time. It is

noted that the search results on *tai256c* benchmark problem (the largest instance of QAP) proves to be much better than that found by Merz in [10]. In [10], the *Average gap* and the best *Gap* of *tai256c* were given as 0.12% and 0.06% respectively. Here, our result shows 0.07% and 0.04% for the *Average gap*

		CPU time	Generation	TG	Average	Average gap	Best	Gap	Success rate	R1	R2	R3	R4
lipa60a	PHGA	79.30	138.80	208.80	108071.00	0.80%	108025	0.75%	0.00%	0.042%	5.02%	-46.31%	-79.38%
107218	SHGA	54.20	186.00	256.00	108116.10	0.84%	107983	0.71%	0.00%				
lipa70a	PHGA	82.10	132.80	202.80	170960.60	0.71%	170886	0.67%	0.00%	0.041%	5.45%	2.53%	-0.84%
169755	SHGA	84.23	139.80	209.80	171030.10	0.75%	170893	0.67%	0.00%				
lipa80a	PHGA	145.20	230.80	300.80	254771.30	0.62%	254663	0.58%	0.00%	0.061%	8.98%	-97.36%	-34.83%
253195	SHGA	73.57	135.50	205.50	254926.80	0.68%	254805	0.64%	0.00%				
lipa90a	PHGA	170.20	232.80	302.80	362707.40	0.58%	362622	0.55%	0.00%	0.038%	6.20%	-14.82%	18.70%
360630	SHGA	148.23	144.40	214.40	362844.80	0.61%	362682	0.57%	0.00%				
lipa60b	PHGA	72.64	118.60	188.60	2843870.40	12.85%	2520135	0.00%	30.00%	4.923%	31.26%	-159.43%	-136.18%
2520135	SHGA	28.00	101.70	171.70	2991112.70	18.69%	2978995	18.21%	0.00%				
lipa70b	PHGA	75.07	119.50	189.50	5305700.60	15.26%	4603200	0.00%	20.00%	1.883%	12.66%	29.25%	24.73%
4603200	SHGA	106.10	131.60	201.60	5407543.60	17.47%	4603200	0.00%	10.00%				
lipa80b	PHGA	78.60	102.90	172.90	9021037.10	16.19%	7763962	0.00%	20.00%	3.388%	20.11%	18.04%	-15.14%
7763962	SHGA	95.90	179.80	242.90	9337411.30	20.27%	9305497	19.86%	0.00%				
lipa90b	PHGA	136.40	134.70	204.70	14533756.70	16.36%	12490441	0.00%	20.00%	3.619%	21.08%	27.83%	35.06%
12490441	SHGA	189.00	114.20	184.20	15079519.10	20.73%	15017837	20.23%	0.00%				

Table 2 Testing results of *lipa* type benchmarks

		CPU time	Generation	TG	Average	Average gap	Best	Gap	Success rate	R1	R2	R3	R4
tai60a	PHGA	70.80	121.70	191.70	7362678.60	2.14%	7317376	1.51%	0.00%	0.493%	19.13%	-152.77%	-135.78%
7208572	SHGA	28.01	108.82	178.82	7399125.45	2.64%	7361202	2.12%	0.00%				
tai80a	PHGA	132.00	209.40	279.40	13793878.40	1.74%	13703252	1.07%	0.00%	0.509%	23.02%	-70.21%	-31.03%
13557864	SHGA	77.55	145.09	215.09	13864464.36	2.26%	13756708	1.47%	0.00%				
tai60b	PHGA	97.80	178.60	248.60	608700699.20	0.08%	608215054	0.00%	40.00%	0.128%	61.63%	-156.02%	-131.51%
608215054	SHGA	38.20	154.80	224.80	609480646.60	0.21%	608352112	0.02%	0.00%				
tai80b	PHGA	118.70	180.10	250.10	822631266.20	0.52%	818720669	0.04%	0.00%	0.484%	48.68%	-27.22%	-21.58%
818415043	SHGA	93.30	169.00	239.00	826629986.30	1.00%	818629320	0.03%	0.00%				
tai64c	PHGA	21.50	0.00	70.00	1855928.00	0.00%	1855928	0.00%	100.00%	0.000%	0.00%	-11.51%	-11.51%
1855928	SHGA	19.28	0.00	70.00	1855928.00	0.00%	1855928	0.00%	100.00%				
tai100a	PHGA	222.80	238.50	308.50	21464686.20	1.61%	21335594	1.00%	0.00%	0.271%	14.68%	33.73%	47.69%
21125314	SHGA	336.21	166.00	243.50	21523065.20	1.88%	21407522	1.34%	0.00%				
tai100b	PHGA	186.90	175.30	245.30	1188882832.20	0.24%	1186007112	0.00%	0.00%	0.144%	37.29%	54.25%	44.53%
1185996137	SHGA	408.50	227.40	297.40	1190599484.50	0.39%	1188431954	0.21%	0.00%				
tai256c	PHGA	10826.00	155.00	225.00	44819951.40	0.07%	44807062	0.04%	0.00%	0.048%	39.86%	37.46%	42.69%
44787190	SHGA	17311.50	136.20	206.20	44841668.80	0.12%	44815296	0.06%	0.00%				

Table 3 Testing results of *tai* type benchmarks

		CPU time	Generation	TG	Average	Average gap	Best	Gap	Success rate	R1	R2	R3	R4
wil100	PHGA	218.00	226.10	292.80	273458.20	0.15%	273236	0.07%	0.00%	0.132%	46.16%	25.87%	35.77%
273038	SHGA	294.09	190.40	253.70	273818.40	0.29%	273340	0.11%	0.00%				
tho150	PHGA	1428.50	308.50	368.20	8158144.60	0.30%	8140370	0.08%	0.00%	0.186%	38.56%	55.17%	62.05%
8133864	SHGA	3186.81	245.40	311.70	8173382.40	0.49%	8149502	0.19%	0.00%				

Table 4 Testing results of *wil100* and *tho150* benchmarks

and best *Gap,* respectively. Hence, this shows that the PHGA is indeed superior on large QAP problem.

Further, Table 4 shows the simulation results on two large QAP problems, *wil100* and *tho150*, where the parallel GA again demonstrates its superiority over the

SHGA on large size QAP problems. The *Average gap* and best *Gap* obtained in [10] were 0.32% and 0.14% respectively while ours were 0.30% and 0.08%. Hence our result for benchmark *tho150* is better than that in [10].

In summary, from Tables 1 to 4, a total of 28 QAP benchmarks were tested, which demonstrate the effectiveness of the parallel GA, especially for large size problems. On the whole, improvement was observed for problems size from 60 onward. The simulation shows that the parallel GA can consistently solve large QAP problems with good solution quality and at a considerably shorter time. Compared with some other results obtained for QAP, the proposed PHGA is competitive.

6. Concluding Remarks

A two-island PHGA implemented in a distributed computing environment, has been studied in the paper. Aiming at improving the efficiency and effectiveness of the algorithm, we present a new algorithm that embeds island PHGA model with NetSolve, a tool based on grid computing technology.

To verify the performance of the PHGA, we run several QAP benchmarks and compare it with the serial GA. The results show that the PHGA offers both improved solution quality and speedup due to parallel processing, especially for the larger size benchmarks, in particular for $n \geq 100$.

From the results, it can be concluded that the PHGA is more powerful for handling large size QAP. This opens up several issues for future research. In particular, the issue on scalability of the parallel GA needs to be studied extensively in a distributed computing framework. While the results reported here focused on the island model GA, there is room to explore the applicability of parallelizing the local search of a serial GA. Along with these, other heuristics for local search can be explored to enhance the performance of the parallel GA.

The successful application of the PHGA within a distributed computing environment in solving the QAP demonstrates its potential in solving other computationally demanding optimization problems. In our opinion the potential of future algorithmic advances for the QAP and other difficult optimization problems can be realized by utilizing the power that computational grids have to offer.

Acknowledgement

The authors would like to thank members in our group for their comments and questions leading to important improvement in this paper.

References

[1] M.H. Lim, Y. Yuan, and S. Omatu, *Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem*, Computational Optimization and Applications, vol. 15,249-268,2000.

[2] M.H. Lim, Y. Yuan, and S. Omatu, *Extensive testing of a hybrid genetic algorithm for quadratic assignment problem*, Computational Optimization and Applications, vol. 23,47-64,2002.

[3] Y.L. Xu, *Hybrid-GA with Stochastic Selection Local Search for Solving Quadratic Assignment Problems*. PhD Confirmation Report, NTU, 2003.

[4] D. Arnold, et al., *User's Guide to NetSolve V1.4, Innovative Computing Laboratory,* Department of Computer Science, University of Tennessee, Knoxville, TN 37996-3450.

[5] T.C. Koopmans and M.J.Beckmann, *Assignment problems and the location of economic activities*. Econometrica 25(1957)53-76.

[6] I. Foster and C. Kesselman (eds.). *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, 1999.

[7] V. Nissen and H. Paul, *A Modification of Threshold Accepting and its Application to the Quadratic Assignment Problem*, OR Spektrum, Vol. 17, pp. 205-210, 1995.

[8] E. D. Taillard, *Comparison of Iterative Searches for the Quadratic Assignment Problem*, Location Science, vol. 3, pp.87-105, 1995.

[9] V. Maniezzo, A. Colorni, and M. Dorigo, *The Ant System Applied to the Quadratic Assignment Problem*, Tech. Rep. 94/28, IRIDIA, University de Bruxelles,1994.

[10] P. Merz and B. Freislenben, *A genetic local search approach to the quadratic assignment problem*, Proc. of Int'l Conf. on Genetic Algorithms (ICGA'97), Morgan Kaufmann: San Mateo, CA, pp. 465-472.

[11] V. Nissen, *Solving the Quadratic Assignment problem with Clues from Nature*, IEEE Transactions on Neural Networks, vol. 5, no. 1, pp. *66*-72, 1994.

[12] Erick Cantu-Paz. *A Survey of Parallel Genetic Algorithms.* Calculateurs Paralleles, Reseaux et Systems Repartis. Vol. 10, No. 2. pp. 141-171, 1998.Paris: Hermes.

[13] Erick Cantu-Paz. *Migration Policies and Takeover Times in Genetic Algorithms*. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.) GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference. (pp. 775). San Francisco, CA: Morgan Kaufmann.

[14] Erick Cantu-Paz, *On the Effects of Migration on the Fitness Distribution of Parallel Evolutionary Algorithms*, In Workshop on Evolutionary Computation and Parallel Processing, GECCO-2000, pp. 3-6.

[15] Braun,H.C. *On Solving Traveling Salesman Problems by Genetic Algorithms*. In Schwefel, H.P. and Manner, R., editors, Parallel Problem Solving from Nature, pp. 129-133, Springer-Verlag, Berlin, Germany, 1990.