International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:1, No:6, 2007

Solving the Quadratic Assignment Problems by a Genetic Algorithm with a New Replacement Strategy

Yongzhong Wu, and Ping Ji

***Abstract*—**This paper proposes a genetic algorithm based on a new replacement strategy to solve the quadratic assignment problems, which are NP-hard. The new replacement strategy aims to improve the performance of the genetic algorithm through well balancing the convergence of the searching process and the diversity of the population. In order to test the performance of the algorithm, the instances in QAPLIB, a quadratic assignment problem library, are tried and the results are compared with those reported in the literature. The performance of the genetic algorithm is promising. The significance is that this genetic algorithm is generic. It does not rely on problem-specific genetic operators, and may be easily applied to various types of combinatorial problems.

***Keywords*—**Quadratic assignment problem, Genetic algorithm, Replacement strategy, QAPLIB.

I. Introduction

THE Quadratic Assignment Problem (QAP) was introduced by Koopmans and Beckmann [1] in 1957 as a mathematical model for the location of indivisible economical activities. QAP is often used to describe a location problem. Let us assign n facilities to n locations with the cost being proportional to the flow between the facilities multiplied with their distances. The objective is to allocate each facility at a location such that the total cost is minimized. Thus we are given two $n \times n$ matrices, the flow matrix $A = (a_{ij})$, and the distance matrix $B = (b_{kl})$. The QAP in Koopmans-Beckmann form can now be written as

$$\min_{\pi \in S_n} C = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{\pi(i)\pi(j)} b_{ij} \qquad (1)$$

where S_n is the set of permutation of $\{1, 2, \ldots, n\}$. Each individual product $a_{\pi(i)\pi(i)}b_{ij}$ is the cost caused by assigning facility $\pi(i)$ to location i and facility $\pi(j)$ to location j. A QAP instance with input matrices A and B is denoted by $QAP(A, B)$ sometimes. If any of the coefficient matrices A, B is symmetric, $QAP(A, B)$ is termed as a symmetric QAP. Otherwise, $QAP(A, B)$ is said to be asymmetric.

In addition to location theory, QAP has other applications such as layout problems, backboard wiring, computer manufacturing, scheduling, process communications, turbine balancing, ranking of archeological data, ranking of a team in a relay race, scheduling parallel production lines. A survey [2] gives extensive references on applications and solution methods for QAP.

It is well known that QAP is NP-hard [3]. Although some "easy" cases are known [4], QAPs in general have been proven to be extremely difficult to be solved to optimality. Most QAP types and instances are collected in the QAPLIB, a well-known library of QAP instances compiled by Burkard et al. [5]. Several famous instances in the QAPLIB, including the problems of size n = 36 posed by Steinberg [6], and problems of size n = 30 posed by Nugent et al. [7], have only been solved to optimality for the first time in 1990s.

II. Literature Review

There are three main exact methods used to find the global optimal solution for a given QAP: dynamic programming, cutting plane techniques, and branch and bound procedures. Research has shown that the latter is the most successful among exact algorithms for solving QAP. Even still, due to the overwhelming complexity of QAP, most problems with their sizes greater than $n = 30$ remain nearly intractable by exact algorithms.

The extreme difficulty of QAP has made it an ideal problem for the development of heuristic search methods. Local search methods, simulated annealing [8], tabu search [9, 10], genetic algorithms [11]-[14], GRASP [15], ant systems [16] and other specialized methods have all been applied to QAP. The performance of different heuristics also tends to vary with certain problem characteristics [17]. Among these heuristics, the tabu search methods, the GRASP[15] and the GA approach [14] are currently the most promising heuristic algorithms to solve QAP.

Conventional genetic algorithms did not find the best known solution for the Nugent's problems of sizes 20 and 30. For larger problems of size up to 100, they seldom really compete

Manuscript received June 15, 2007. This work was supported by the project of the Hong Kong Polytechnic University (No. A-PG05).

Yongzhong Wu is with the Department of Industrial and Systems Engineering, The Hong Kong Polytechnic University, Hong Kong. (corresponding author to provide phone: +852-27666630; E-mail address: w.yz@polyu.edu.hk).

Ping Ji is with the Department of Industrial and Systems Engineering, The Hong Kong Polytechnic University, Hong Kong (e-mail: pji@polyu.edu.hk).

with tabu search procedures. In 2000, Ahuja, Orlin and Tiwari [13] obtained very promising results on large scale QAPs in QAPLIB by applying a version of GA called a greedy genetic algorithm. Recently, Drezner [14] designed a new GA with a problem-specific crossover rule and a tabu search, and obtained even better results than those obtained by Ahuja et al. Currently, this new genetic algorithm seems to be the best heuristics to solve QAPs in terms of accuracy [14].

The genetic algorithm proposed by Drezner exploited the problem-specific characteristics in designing the crossover operator, which increased the complexity of the algorithm and made it difficult to be used to solve other problems.

In this paper, a genetic algorithm based on a new replacement strategy is devised to solve the QAPs. The purpose is to examine the probability of devising a heuristic for solving QAPs efficiently without using any problem-specific characteristics. The general-purpose GA may have the potential to solve other types of NP-hard problems.

III. The Proposed Genetic Algorithm

A. The New Replacement Strategy

The replacement strategy refers to a selection strategy defining how to select the next generation members from the offspring and the last generation members. It is very important particularly for highly-constraint problems because it guides the searching of the algorithm throughout the searching space and thus influences the performance of the algorithm.

The most commonly used replacement strategy in the literature is the steady-state replacement strategy. In every generation, individuals are selected for conducting the genetic operators. Every new offspring will be compared with the worst member in the population. If the offspring is better than the worst member, then the offspring will replace it.

A new replacement strategy is proposed in this paper. It aims to improve the global searching ability of the algorithm. It incorporates two different replacement policies, *i.e.*, the replace-worst policy and the replace-parent policy. The replace-parent policy is taken once every generation, *i.e.*, every new offspring only compares with its parent (the chromosome before mutation). If the fitness of the offspring is better than its parent, then the new offspring will replace it. In comparison, the replace-worst policy is taken once in every certain amount of generations, *i.e.*, each new offspring will compare with the worst chromosome in the current population. If the new offspring is better, then it will replace this worst member. By adjusting the frequency of the replace-worst policy, the convergence of the searching process and the diversity of the population can be well balanced, and the potential of each chromosome can be well exploited before it is removed out of the population.

The scheme of the genetic algorithms based on this new replacement strategy can be described as follows. In the scheme, the replace-worst policy is applied once in every T_c generations (T_c is called the period for the replace-worst policy).

begin
  create initial population;
  for every generation, repeat
    randomly select individuals into the mating pool;
    apply genetic operators to generate offspring;
    (maybe) apply post-crossover heuristic on offspring;
    if not in $i \times T_c$ generations
      compare every offspring with its similar parent and remove the worse one;
    else
      compare every offspring with the worst member in the current population and remove the worse one
  until some stopping criterion is met
end;

B. Description of the Proposed GA for the QAP

In the following, the proposed GA for the QAP is described according to the chromosome representation, the fitness function, the crossover, and the post crossover heuristic.

Chromosome Representation

In the genetic algorithm for the QAP, permutation representation is employed, which is illustrated in Fig. 1. The illustration takes the facility location problem as an example.

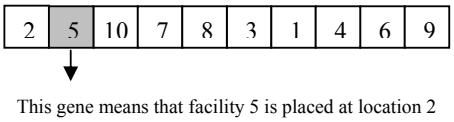

Fig. 1 Representation scheme of the genetic algorithm

In the above representation, the value of every gene represents the facility that is assigned to corresponding location. In the chromosome shown in Fig. 1, there are 10 facilities to be placed at 10 locations. For example, the second gene in the chromosome means that facility 5 is placed at location 2.

Objective and Fitness Functions

The objective of the genetic algorithm is to minimize the total cost C as described in the objective function (1). The fitness function for the chromosome in the genetic algorithm is defined as:

$$f_i = 1 / C_i \tag{2}$$

where f_i is the fitness of chromosome i, and C_i is the objective value of chromosome i.

Crossover

A special type of uniform crossover with constant percentage of exchanging genes. This percentage of exchanging genes is set to be small value like 20%, so each of the two children will be much similar to one of their parents. This crossover process is illustrated in Fig. 2. After the

crossover, individuals A and B produce C and D. We define A as the direct parent of C since they are similar to each other, and likewise, we define B as the direct parent of D.

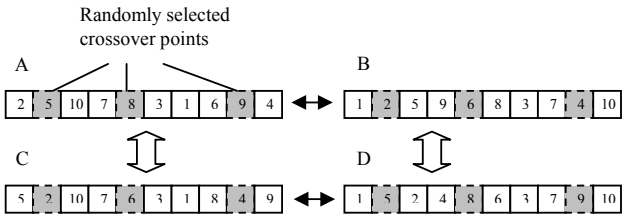

Fig. 2 Crossover scheme in the new genetic algorithm

In every generation, every chromosome is paired up with another chromosome randomly. Crossover operator will be applied to all pairs of chromosomes. That means the crossover rate is 1.

Through preliminary tests, it was found that the mutation operator did not have much impact on the performance of the GA for solving the problem, so there is no mutation operator used in the proposed GA.

The Post-Crossover Heuristic

To help improve the evolution of all the chromosomes, a post-crossover heuristic is applied to the newborn offspring. The combined heuristic used in our genetic algorithm is the descent local search heuristic, which is defined as follows:

Step 1 Examine the change in the fitness value for all the pairwise exchanges of nearby alleles.

Step 2 The best improving exchange is executed and go back to Step 1 again.

Step 3 If no improving exchange is found, the heuristic terminates.

IV. Computational Experiments

In this section, instances in the QAPLIB [5] were used to test the new genetic algorithm. The results were compared with those obtained by Ahuja et al. [13] and those obtained by Drezner [14]. The effect of the new replacement strategy will also be evaluated.

The population size in the genetic algorithm was set to be 100. The period for the replace-worst policy T_c was set to be equal to the problem size n, i.e., the replace-worst policy will apply once every n generations. The percentage of exchanging genes in the crossover operator was set to be 0.2. The algorithm was set to be run until there was no improvement during n generations.

A. Computation Results

In this section, the new genetic algorithm was tested on all instances of size n ranging from 30 to 100, most of which still can not be solved to optimality. The program was coded in Microsoft Visual C++ 6.0, and ran on a desktop computer with Pentium III 866 CPU and 256 RAM, which was similar to the computer used by Ahuja et al. [13]. Ahuja et al. ran the algorithm only once for each problem. Drezner ran his algorithm 200 times for each problem, and we ran our genetic algorithm 20 times for each problem. The results shown in Table I are the averages. The numeric digits in a problem name state the problem size. The percentage deviations from the best known solutions are given in the "Gap" columns.

TABLE I
Comparison of our GA and and the Other Two GAs

Problem	*	Our GA		Ahuja et al		Drezner	
		Gap (%)	Time (min)	Gap (%)	Time (min)	Gap (%)	Time (min)
esc32a	20	0	0.37	0	6.36	0	0.35
esc32b	20	0	0.24	0	6.67	0	0.30
esc32c	20	0	0.20	0	6.49	0	0.27
esc32d	20	0	0.15	0	5.88	0	0.28
esc32e	20	0	0.23	0	6.16	/	/
esc32f	20	0	0.17	0	6.14	/	/
esc32g	20	0	0.33	0	6.18	/	/
esc32h	20	0	0.26	0	5.82	0	0.29
esc64a	20	0	4.13	0	43.85	/	/
kra30a	20	0	0.29	0	5.02	0	0.33
kra30b	20	0	0.34	0	5.51	0	0.33
lipa30a	20	0	0.16	0	5.74	/	/
lipa30b	20	0	0.16	0	5.62	/	/
lipa40a	20	0	1.02	0.960	17.03	/	/
lipa40b	20	0	0.90	0	17.10	/	/
lipa50a	20	0	3.31	0.950	24.77	/	/
lipa50b	20	0	2.98	0	25.14	/	/
lipa60a	20	0	10.35	0.770	50.95	/	/
lipa60b	20	0	9.97	0	50.79	/	/
lipa70a	6	0.127	30.08	0.710	102.47	/	/
lipa70b	20	0	28.23	0	102.05	/	/
lipa80a	2	0.242	47.82	0.610	158.55	/	/
lipa80b	20	0	45.25	0	158.31	/	/
lipa90a	1	0.187	61.88	0.580	205.97	/	/
lipa90b	20	0	60.08	0	205.32	/	/
nug30	20	0	0.36	0.070	5.9033	0	0.37
sko42	20	0	1.57	0.250	16.77	0	1.15
sko49	10	0.038	3.78	0.210	20.87	0.009	2.13
sko56	20	0	7.36	0.020	49.6	0.001	3.24
sko64	20	0	12.11	0.220	63.14	0	5.85
sko72	3	0.042	35.39	0.290	84.63	0.014	8.36
sko81	2	0.067	57.28	0.200	182.74	0.014	13.30
sko90	1	0.073	102.50	0.270	211.63	0.011	22.35
sko100a	2	0.051	174.13	0.210	276.80	0.018	33.55
sko100b	7	0.039	165.50	0.140	245.49	0.011	34.05
sko100c	16	0.015	158.54	0.200	338.57	0.003	33.80
sko100d	11	0.022	184.41	0.170	338.37	0.049	33.90
sko100e	10	0.030	167.31	0.240	352.12	0.002	30.67
sko100f	5	0.017	170.88	0.290	357.98	0.032	35.74
ste36a	18	0.025	0.84	0.270	11.827	0.005	0.55
tho30	20	0	0.31	0	6.59	0	0.35
tho40	9	0.041	1.98	0.32	15.97	0.010	0.98
wil50	4	0.028	5.06	0.070	35.25	0.002	1.99
wil100	2	0.041	176.28	0.200	342.40	0.002	33.11

Notes:
*:Number of times out of 20 runs that the best known solutions obtained.
Gap (%) : Percentage deviations over the best known solutions
Time(min): The average running time for a single run of the algorithm.
The results by Ahuja et al (2000) were obtained by GA-3, which was their best algorithm

There are totally 44 instances for the comparison. It can be seen that our genetic algorithm performed much better than the greedy genetic algorithm designed by Ahuja et al. [13], in terms of both objective value and computation time. The average percentage deviation from the best known solution for our genetic algorithm was 0.025%, while that for Ahuja's GA is 0.187%. And the computation time for our genetic algorithm was much less than Ahuja's GA (GA-3). For all these 44 problems, our genetic algorithm found the best known solution at least once in 20 runs.

We can also see that our genetic algorithm performed as good as the genetic algorithm designed by Drezner [14] when the problem size is not so large. But Drezner's algorithm seems to perform slightly better and faster when the problem size is larger than 50. This is partly because Drezner's algorithm was designed only for symmetric problems. As stated by Drezner [14], this may significantly reduce the computation time of the fitness function. But the main reason was that it incorporated a problem-specific which exploit the characteristics of QAPs and a highly efficient tabu search into the genetic algorithm. In contrast, there are only two main differences between our genetic algorithm and conventional genetic algorithms: the new replacement strategy and the descent local search heuristic to improve the offspring, both of which are not problem-specific. So the our GA can be easily used for solving other types of problems.

B. Evaluation of the Effect of the New Replacement Strategy

In this section, in order to examine the effect of the new replacement mechanism, the post-crossover heuristic was removed. The results were compared with those obtained by the conventional genetic algorithm devised by Tate and Smith [12].

The population size of our GA was set to be 100, the same as that in the GA proposed by Tate and Smith. We ran our genetic algorithm for 2000 generations. Other parameters were set to be the same as those in Section 4A.

The comparison of these two algorithms is shown in Table II. Because Tate and Smith only tested two instances of sizes larger than 30, we compared the results of these two instances. From the results, it can be seen that even without the post-crossover heuristic, our genetic algorithm obtained much better results than the genetic algorithm proposed by Tate and Smith.

TABLE II
COMPARISON BETWEEN THE STANDARD GA USED BY TATE AND SMITH (1995) AND OUR GA WITHOUT POST-CROSSOVER HEURISTIC

Problem	Best	GA by Tate and Smith			Our GA		
	Known	Best	Average	gap (%)	Best	Average	gap (%)
nug30	6124	6184	6305.4	2.96	6124	6176.1	0.85
ste36c	8239.11	8592	8946.2	8.58	8239.11	8354.46	1.4

In order to evaluate the effect of the value of T_c, the period for the replace-worst policy, on the performance of the proposed genetic algorithm, we further varied the values of T_c, and compared the results obtained by the genetic algorithm with different T_c. In this test, the post-crossover heuristic was also removed. The instance nug30 was used as the testing instance.

T_c was set to be different values of 10, 20, 30, 40, 50, and 60. The genetic algorithm ran 20 times for each T_c value. In each run, the genetic algorithm was running for 1000 generations. Other parameters were set to be the same as those in Section 4A. Table III shows the average results obtained by the genetic algorithm with different T_c values.

TABLE III
AVERAGE RESULTS OBTAINED BY THE GENETIC ALGORITHM WITH DIFFERENT T_c

T_c Value	10	20	30	40	50	60
Average value	6312	6293	6258	6237	6190	6176

From Table III, it can be seen that the period for the replace-worst policy T_c has significant effect on the performance of the genetic algorithm. It should be noted that the optimal solution to nug30 is 6124. The genetic algorithm could obtain very good result when T_c =60. Fig. 3 illustrates the convergence process of the genetic algorithm with T_c =10 and T_c =60. From the figure, we can see that when T_c is small, the genetic algorithm tends to converge in the early stage. When T_c is large, better solution may be found. However, the computation effort will be increased. The CPU time for the genetic algorithm with T_c =60 was about 7 seconds. When the problem size increases, the computation effort of the algorithm will increase exponentially. Fine tuning of the T_c value is important to the overall performance of the genetic algorithm.

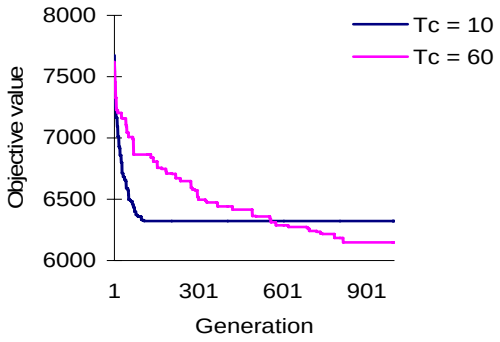

Fig. 3 Convergence process for T_c =10 and T_c =60

V. DISCUSSION AND CONCLUSION

This paper proposed a new genetic algorithm for the quadratic assignment problems, which are NP-hard. The new genetic algorithm is based on a new replacement strategy, which aims to balance the exploitation and exploration of the searching space. The test on the QAP instances obtained

promising results. The significance is that the new genetic algorithm is not problem specific, which means the algorithm does not rely on a problem-specific characteristics and can be easily applied to various optimization problems.

However, it would be quite difficult to judge about the performance of an algorithm based on a single type of optimization problems. Other types of problems shall be tested in the future to evaluate the proposed genetic algorithm and the new replacement strategy.

REFERENCES

[1] T. C. Koopmans, M. J. Beckmann, Assignment problems and the location of economic activities, Econometrica, 25(1957), pp.53-76.
[2] R. E. Burkard, E. Cela, P. M. Pardalos, L. S. Pitsoulis, The quadratic assignment problem, in: Anonymous Handbook of Combinatorial Optimization, Volume 3, Kluwer, 1998, pp.241-337.
[3] S. Sahni, T. Gonzalez, NP-complete approximation problems, Journal of the Association for Computing Machinery, 23(1976), pp.555-565.
[4] R. E. Burkard, E. Cela, G. Rote, G. J. Woeginger, The quadratic assignment problem with a monotone anti-Monge and a symmetric Toeplitz matrix: easy and hard cases, Mathematical Programming, 82(1998), pp.125-158.
[5] R. E. Burkard, S. E. Karisch, F. Rendl, QAPLIB: A quadratic assignment problem library, Journal of Global Optimization, 10(1997), pp.391-403.
[6] L. Steinberg, The backboard wiring problem: A placement algorithm, SIAM Review, 3(1961), pp.37-50.
[7] C. E. Nugent, T. E. Vollman, J. Ruml, An experimental comparison of techniques for the assignment of facilities to locations, Operations Research, 16(1968), pp.150-173.
[8] D. T. Conolly, An improved annealing mechanism for the QAP, European Journal of Operational Research, 46(1990), pp.93-100.
[9] J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem, ORSA Journal on Computing, 2(1990), pp.33-45.
[10] E. D. Taillard, Robust tabu search for the quadratic assignment problem, Parallel Computing, 17(1991), pp.443-455.
[11] C. Fleurent, J. Ferland, Genetic hybrids for the quadratic assignment problem, in: Anonymous DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1994, pp.173-187.
[12] D. D. Tate, A. E. Smith, A genetic approach to the quadratic assignment problem, Computers and Operations Research, 1(1995), pp.855-865.
[13] R. K. Ahuja, J. B. Orlin, A. Tiwari, A greedy genetic algorithm for the quadratic assignment problem, Computers and Operations Research, 27(2000), pp.917-934.
[14] Z. Drezner, A new genetic algorithm for the quadratic assignment problem, INFORMS Journal on Computing, 15(2002), pp.320-330.
[15] Y. Li, P. M. Pardalos, Resende, M. G. C., A greedy randomized adaptive search procedure for the quadratic assignment problem. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 16(1994), pp.237-261.
[16] L. M. Gambardella, E. D. Taillard, M. Dorigo, Ant colonies for the quadratic assignment problem, Journal of the Operational Research Society, 50(1999), pp.167-176.
[17] E. D. Taillard, Comparison of iterative searches for the quadratic assignment problem, Location Science, 3(1995), pp.87-105.