Parallel Optimization with Mutation Operator for The Quadratic Assignment Problem

Tansel Dokeroglu, Umut Tosun, Ahmet Cosar

METU Computer Engineering Department Ankara / TURKEY
{tansel,tosun,cosar}@ceng.metu.edu.tr

Abstract. Recombination operator crossover is a tool that is widely used to obtain near optimal solutions for larger problem instances of the Quadratic Assignment Problem (*QAP*), whereas, the mutation operator is the less preferred tool. However, due to its low disruptive nature the mutation operator does have a great potential to solve the *QAP*. In this study we compared the performance of the mutation operator with the well-known crossover operators that are designed for use with the *QAP*. The mutation operator works by swapping values iteratively and outperforms most of the well-known crossovers. We conclude that, as reported to date, the mutation operator can provide either the optimum or very close to the best available solution for problem instances in the *QAP* library.

Keywords: Quadratic Assignment Problem, Parallel Genetic Algorithms, Mutation, Crossover.

1 Introduction

The quadratic assignment problem is a well-known NP-hard combinatorial optimization problem first introduced by Koopmans and Beckmann [1,2] as a mathematical model for the location of indivisible economic activities. Since this introduction it has become one of the most interesting challenges for scientists to use the model in a variety of problems. Some of the areas that have been successfully modeled as a QAP are; typewriter keyboard design, backboard wiring [3], layout design [4], turbine balancing [5], scheduling [6,7,8], data allocation [9], assigning gates to arriving and departing flights at an airport application [10] and assigning rooms to people with neighborhood constraints [11]. Furthermore, service allocation problem with the purpose of minimizing the container rehandling operations at a shipyard [12], the travelling salesman, bin-packing, maximum clique, linear ordering, and graph-partitioning problems are among the interesting application areas of the QAP.

Applying QAP allows the assignment of n facilities to n locations at a cost proportional to the amount of material flow between the facilities multiplied by the distances between the locations, plus the costs for placing the facilities at their respective locations. The objective is to find an allocation that minimizes the total cost of allocating and operating all facilities. QAP can be formally modeled using three n $\times$ n matrices, A, B, and C as follows:

A= (a_{ik}), a_{ik} is the flow amount from facility i to facility k.

B= (b_{jl}), b_{jl} is the distance from location j to location l.

C= (c_{ij}), c_{ij} is the cost of placing facility i at location j.

The Koopmans-Beckmann form of QAP can be written as:

$$\min_{\varphi \in S_n}\left(\sum_{i=l}^{n}\sum_{k=1}^{n} a_{ik} b_{\varphi(i)\varphi(k)} + \sum_{i=1}^{n} c_{i\varphi(i)}\right)$$

Where S_n is the set of all permutations of integers 1,2,…,n. Each individual product $a_{ik}b_{\varphi(i)\varphi(k)}$ is the transportation cost resulting from assigning facility i to location $\varphi(i)$ and facility k to location $\varphi(k)$. Each term $c_{i\varphi(i)}$ + $\sum_{k=1}^{n} a_{ik} b_{\varphi(i)\varphi(k)}$ is the total cost given, for facility multiplied by the cost for installing it at location $\varphi(i)$, plus the cost of transportation to all other facilities k, installed at locations $\varphi(1)$, $\varphi(2)$,…, $\varphi(n)$. An instance of QAP with input matrices A, B, and C is denoted as QAP(A,B,C). If there is no C term, then it is written QAP (A,B). Lawler [13] introduced a four-index cost array D= (d_{ijkl}) instead of the three matrices and obtained the general form of the QAP;

$$\min_{\varphi \in S_n} \sum_{i=1}^{n}\sum_{k=1}^{n} d_{i\varphi(i)k\varphi(k)}$$

The relationship of this formula with the Koopmans-Beckmann problem is;

$d_{ijkl} = a_{ik} b_{il}$ (i,j,k,l = 1,2,…,n; i $\neq$ k or j $\neq$ l);

$d_{ijij} = a_{ii} b_{jj} + c_{ij}$ (i,j=1,2,…,n)

Since they were introduced by Holland [14] Genetic Algorithms (GAs) have been frequently used to solve search and optimization problems. GAs use a computational model that simulates the natural processes of selection and evolution. Individuals with better qualities have a greater probability of surviving thus, to reproduce and transmit their genetic characteristics to future generations. Each potential solution existing in the search space is considered as an individual (phenotype) and represented by strings called chromosomes. Genes are the atomic parts of chromosomes codifying a specific characteristic of the chromosome. There are several ways to encode individuals for a variety of applications; a random population is generated in the first stage of the algorithm and then selection, crossover, and mutation operations are applied cyclically, thus the GAs create new generations [15]. The individual having the best fitness value in the population is the solution of the problem.

The main point of our study is that crossover operators and the mutation operator generate new individuals by recombining the characteristics of the parents. For each pair of individuals, the parent chromosomes are split into parts and genes are exchanged to generate new chromosomes. Individuals not subjected to any operation are copied into the next generation. If a mutation occurs, it modifies an individual’s genetic code [16] (Figure 1).

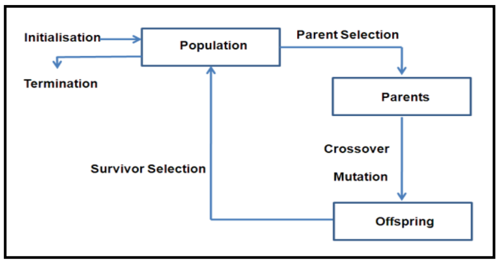

Fig. 1. Dataflow diagram of a genetic algorithm.

This study presents the analysis that we undertook using four different well-known crossovers and the mutation operator [17,24,25]. All the tests were performed with an Island Parallel GA (IPGA) during the tests to evaluate the performance of the operators. We performed comprehensive experiments and reported the performance of crossovers and the mutation operator. In this paper first we give a formal description of the QAP and a brief overview of general strategies of GAs. In section 2 the related crossover operators and the mutation operator are explained. Section 3 explains the IPGA. The environment and the test results obtained with different instances of the QAP library for all operators are discussed in section 4. Finally, section 5 presents our concluding remarks.

2 Crossover and the mutation operators

The structure of all the chromosomes analyzed in this study is shown in Figure 2.

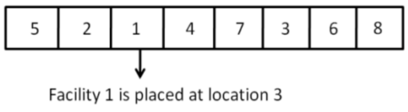

Fig. 2. Representation scheme of the crossover operator for the *QAP*.

Order Based Crossover (OBX) preserves the relative order of genes in the chromosomes. A number of items are selected from one of the parents and copied to the offspring. The other genes are copied from the second parent in the order given previously. OBX can be successfully applied to scheduling problems [18].
The Multiple Crossover (MPX) was developed by Misevičius [20] and it is possible to use more than two parents with MPX so that the offspring derives information from

many parents. The selection of genes depends on their probability of existence compared with the other genes.

Let p_1 and p_2 be two parents, the Swap Path Crossover (SPX) either starts from a random number or the leftmost element of the parents until all the genes are examined [21]. If the genes are the same, move to the next position; else perform a swap of the two genes in p_1 and p_2, so that the current positions of the parents become the same. The child with better fitness value takes the place of its parent. The genes in the two resulting children chromosomes are then considered, starting from the next position and so on. The best solution obtained serves as an offspring. In 2003 Drezner introduced an original and efficient operator, Cohesive Crossover (COHX) [22]. A pair of parents is randomly selected from the population. The parent with a better fitness value considered to be the first parent. A median distance value is evaluated for a chosen pivot value for the first parent and a distance matrix is constructed. Sites closer than the median value to the pivot value are assigned from the first parent. All other sites are assigned from parent 2 [22]. Mutation for the QAP is a simple operator that interchanges two genes. More than one interchange can be applied to the chromosome. The genes can be selected randomly or in a different way to produce new chromosomes (Figure 3). In our mutation operator, after a random swap is executed the new fitness value is evaluated.

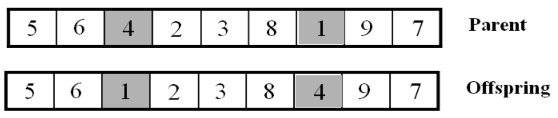

Fig. 3. Mutation

Operators can be analyzed according to different parameters such as the number of parents, execution time and the space complexity. The mutation works on a single parent, the rest of the operators work with two parents whereas the MPX uses more than two parents. The production time of the new offspring is mostly $O(n^2)$, where n is the size of the chromosome. Only MPX has $O(n^2)$ space complexity, and all the other operators have $O(n)$ space complexities. The fitness evaluation is the same for all the operators.

3 Island parallel genetic algorithm

Parallel computation allows many calculations to be performed simultaneously [18,19]. The granularity of the genetic algorithms (GAs) makes them very compatible with this principle of the parallel computers. The divisibility of the population and the computation of the fitness values of the individuals in parallel are the main motivation for the parallelization of GAs. An IPGA is an extension of a GA. Its advantage is to facilitate the evolution of different subpopulations in diverse directions concurrently, dramatically speeding up the search process, thus potentially producing higher quality solutions by creating a larger number of individuals and generations [18]. The IPGA used in this work contains as many subpopulations as the number of the slave processors, and these subpopulations can exchange individuals at the initial steps of search until a 10% increase is achieved in the total population. The migration of best individuals among processors is expected to improve the average quality of the subpopulations, while the increased communication cost between processors and larger population size are the negative effects. Each subpopulation is assigned to a slave processor on which genetic operations are performed independently of the other processors [23]. Non-redundant initial populations are used at each slave node. The detailed flow of the IPGA is given in Algorithm 1.

IPGA terminates its execution when the slaves reach the time limit defined by the master processor. The time based termination condition is very suitable for PGAs, so parallelizing the entire slave nodes are selected using this method. A truncation selection is used to start with a qualified population. After the generation of the initial population, the best performing 1/10 of the population are selected to survive by truncation. During the crossover and the mutation, randomly selected individuals are used and if the offspring are better, they are swapped with the worst performing individual in the population.

Algorithm 1 *IPGA Algorithm*

```
If Node is the Master
     Int itr=1;
  While itr $\leq$ Number of Slaves
       Begin
           Receive the Best Results from Slave (itr);
           Update the Best Configuration;
           Itr++;
       End
     Show the Best Configuration;
Endif
If Node is a Slave
     Double Start_time, End_time, Time_Elapsed;
     Take Start_time;
     P $\leftarrow$ Generate non-redundant Population;
  While Time_Elapsed $\leq$ Termination Limit
       Begin
           Send the Best Solution to Other Slaves;
           Receive the Best Individual From Other Slaves;
           (p1,p2) $\leftarrow$ SelectParentsPair (P);
           s $\leftarrow$ Crossover (p1,p2);
           P $\leftarrow$ PopulationUpdate (P,s);
           Take End_time;
           Time_Elapsed= End_time-Start_time;
       End
     Send the Best Solution to the Master;
Endif
```

4 Tests and computational results

In the experiments, the same execution time, number of processors and initial populations were provided for all the operators. Two experiment sets were executed as short and long tests. The population size before truncation was fixed with 1,000 individuals at each slave for all tests. The total number of individuals at all slaves after truncation is 2,000, as the truncation ratio is 10%. The crossovers and the mutation operator were tested with 39 well-known different data sets of the QAPLIB [26]. The QAPLIB serves as a library of benchmark problem instances for the researchers working on the QAP. Most of the proposed algorithms for solving the QAP have been tested on these benchmark instances. There are more than 100 instances that have been derived either from randomly generated problem instances or real life applications. The hospital layout (kra30), Manhattan distances of rectangular grids (had12), and the backboard wiring (ste36a) are some of the problem instances that can be found in QAPLIB. Our experiments were performed on a cluster computer with 20 slave processors, and one master. A cluster is a group of loosely coupled computers working together closely, so that in many aspects they can be thought as a single computer. The complete configuration of our HPC system comprised; 46 nodes, each with 2 CPUs giving 92 CPUs. Each CPU had 4 cores giving a total of 368 cores. Each node had 16GB of RAM giving 736 GB of total memory, and 146GBs of disk storage per node, giving a total disk space of 6.5TB. In addition, there was a total of 6TB common storage derived from 2 high performance disk units each with 3 TB. To provide high-bandwidth communication among HPC nodes, two 24 port Gigabit ethernet switches, and one 24 port high performance switch were used. The software installed on the HPC was; a Scientific Linux v4.5 64-bit operating system, Lustre v1.6.4.2 parallel file system, Torque v2.1.9 resource manager, Maui v3.2.6 job scheduler, and an Open MPI v1.2.4.

In developing the IPGA, C++ programming language and Message Passing Libraries (MPI) were used. We ran each test 10 times and took the average of the best results received by the master processor. Short tests were run over 20 clock seconds (400 CPU seconds) and long tests over 500 clock seconds (10,000 CPU seconds) thus the longer tests are 25 times larger. The results of the comparisons are shown in Tables 1 and 2.

Table 1. Short run test results of 39 instances from the *QAPLIB*. Each column of the operators gives the percentage deviation of the operators from the BKS. (BKS : Best Known Solution).

Instance	*BKS*	*OBX*	*MPX*	*SPX*	*COHX*	*Mutation*
sko42	*15812*	*2.800*	*16.290*	*2.530*	*2.500*	***0.660***
sko49	*23386*	*2.570*	*15.340*	*3.000*	*4.120*	***1.540***
sko56	*34458*	*3.440*	*16.310*	*4.390*	*6.810*	***2.880***
sko64	*48498*	*3.430*	*14.250*	*7.460*	*8.770*	***3.500***
sko72	*66256*	*5.110*	*14.040*	*8.980*	*10.840*	***4.420***
sko81	*90998*	*5.990*	*13.560*	*9.790*	*11.140*	***5.030***
sko90	*115534*	*5.870*	*14.480*	*9.490*	*11.870*	***5.530***
sko100a	*152002*	*6.780*	*13.320*	*10.320*	*12.110*	***6.200***

sko100b	*153890*	*7.960*	*13.630*	*10.270*	*11.590*	***6.140***
sko100c	*147862*	*7.950*	*14.000*	*10.370*	*12.460*	***6.580***
sko100d	*149576*	*8.670*	*13.690*	*10.630*	*11.630*	***6.150***
sko100e	*149150*	*8.500*	*14.480*	*10.990*	*12.490*	***6.690***
sko100f	*149036*	*8.540*	*13.610*	*10.370*	*12.000*	***6.230***
	Average	*5.970*	*14.385*	*8.353*	*9.872*	***4,734***
tai20a	*703482*	*6.020*	*11.070*	*3.200*	*1.440*	***0.860***
tai25b	*1167256*	*5.740*	*9.510*	*3.540*	*2.820*	***1.180***
tai30a	*1818146*	*7.930*	*11.340*	*5.290*	*3.870*	***1.090***
tai35a	*2422002*	*8.930*	*13.250*	*6.190*	*3.600*	***1.230***
tai40a	*3139370*	*9.200*	*13.840*	*6.200*	*3.660*	***2.340***
tai50a	*4938796*	*11.070*	*14.330*	*5.280*	*7.240*	***4.150***
tai60a	*7205962*	*10.120*	*13.690*	*7.530*	*10.760*	***5.210***
tai80a	*13515450*	*9.380*	*12.830*	*9.690*	*11.190*	***6.230***
tai100a	*21059006*	*9.570*	*12.140*	*9.830*	*11.200*	***7.150***
	Average	*8.662*	*12.444*	*6.306*	*6.198*	***3.271***
esc32a	*130*	*44.620*	*92.310*	*16.920*	*12.310*	***0.000***
esc32b	*168*	*28.570*	*90.480*	*23.810*	*14.290*	***0.000***
esc32c	*642*	*0.000*	*11.210*	***0.000***	***0.000***	***0.000***
esc32d	*200*	*8.000*	*27.000*	*3.000*	***0.000***	***0.000***
esc32e	*2*	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***
esc32f	*2*	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***
esc32g	*6*	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***
esc32h	*438*	*5.480*	*19.180*	***0.000***	***0.000***	***0.000***
esc64a	*116*	*8.620*	*56.900*	*3.450*	*3.450*	***0.000***
esc128	*64*	*200.000*	*237.500*	*156.250*	*187.500*	***31.250***
	Average	*29.529*	*53.458*	*20.343*	*21.755*	***3.125***
nug20	*2570*	*1.950*	*10.970*	*0.700*	*0.160*	***0.000***
nug21	*2438*	*1.380*	*10.360*	*1.480*	*1.150*	***0.000***
nug22	*3596*	*1.280*	*11.900*	*1.060*	*0.720*	***0.000***
nug24	*3488*	*3.100*	*15.080*	*1.260*	*1.610*	***0.000***
nug25	*3744*	*1.660*	*14.740*	*1.870*	*0.050*	***0.000***
nug27	*5234*	*5.780*	*15.900*	*2.030*	*1.220*	***0.000***
nug30	*6124*	*3.170*	*14.730*	*1.470*	*1.600*	***0.000***
	Average	*2.142*	*13.383*	*1.310*	*0.930*	***0.000***
Overall Average		*11,576*	*23,418*	*9,078*	*9,689*	***2,880***

Table 2. Long run test results of 39 instances. Each column of the operators gives the percentage deviation of the operators from the BKS.

Instance	***OBX***	***MPX***	***SPX***	***COHX***	***Mutation***
sko42	*2.800*	*14.000*	*2.530*	*2.500*	***0.290***
sko49	*3.090*	*13.280*	*3.010*	*2.100*	***0.220***
sko56	*4.000*	*15.180*	*3.220*	*3.150*	***0.740***

sko64	3.920	14.260	3.400	3.240	**0.510**
sko72	4.680	13.520	2.960	2.940	**0.130**
sko81	4.170	13.850	2.860	3.110	**0.460**
sko90	4.320	13.850	3.130	3.800	**0.860**
sko100a	3.730	13.020	3.410	3.210	**1.120**
sko100b	4.560	12.820	2.880	3.420	**1.090**
sko100c	4.980	13.540	3.100	3.190	**0.880**
sko100d	4.020	13.280	2.970	3.570	**0.780**
sko100e	5.040	13.520	3.380	3.490	**1.250**
sko100f	4.280	13.010	3.010	3.460	**1.220**
Average	4.122	13.625	3.066	3.168	**0.735**
tai20a	5.430	9.960	2.890	2.120	**0.530**
tai25b	4.990	9.900	3.990	1.730	**1.380**
tai30a	7.830	10.760	4.720	3.070	**1.280**
tai35a	10.170	12.170	4.740	3.510	**1.200**
tai40a	8.480	12.210	5.720	4.590	**1.850**
tai50a	10.920	12.880	6.740	5.400	**2.190**
tai60a	10.110	13.370	6.490	4.950	**1.820**
tai80a	9.850	12.390	6.390	5.360	**2.320**
tai100a	9.950	11.810	6.270	4.960	**2.980**
Average	8.637	11.717	5.328	3.966	**1.728**
esc32a	53.850	84.620	20.000	12.310	**4.620**
esc32b	38.100	80.950	23.810	14.290	**0.000**
esc32c	**0.000**	7.170	**0.000**	**0.000**	**0.000**
esc32d	4.000	23.000	2.000	**0.000**	**0.000**
esc32e	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
esc32f	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
esc32g	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
esc32h	5.020	15.070	0.910	**0.000**	**0.000**
esc64a	6.900	44.830	**0.000**	**0.000**	**0.000**
esc128	78.120	215.620	15.620	6.250	**0.000**
Average	8.637	11.717	5.328	3.285	**0.462**
nug20	2.330	9.260	0.320	1.110	**0.000**
nug21	1.210	10.010	1.090	0.000	**0.000**
nug22	1.610	10.180	0.500	0.500	**0.000**
nug24	3.210	13.020	2.580	0.460	**0.000**
nug25	1.120	12.130	0.960	0.048	**0.000**
nug27	6.840	13.110	1.990	1.110	**0.000**
nug30	3.360	14.240	1.400	0.980	**0.000**
Average	2.811	11.707	1.263	0.601	**0.000**
Overall Average	6,052	12,192	3,746	2,755	**0,731**

MPX is the worst performing operator for all the short run tests but the mutation outperforms all the operators therefore the order of the performance in the short run tests from high to lower is; mutation, SPX, COHX, OBX, and MPX. In the long run tests, we can see that given more time, all the operators can provide better results, however, the MPX is once again the worst performing operator. Given more execution time, both COHX and SPX improve their solution qualities drastically when compared with other operators. As can be seen from the results, the crossover operators cannot produce solutions that are as good as the mutation operator. This shows that they are more like diversification search tools, whereas the mutation operator seems to resemble an intensification search tool. As a result, less disruptive operators perform better. It is also observed that in different instances, crossovers can lead their search directions to diverse spaces. Figure 4 shows a ranking of the operators according to all the average best result deviations of the short and long tests. The mutation operator, SPX, and COHX are the best performing operators below the average of 5% deviation from the best known solutions. Table 3 shows the results of the mutation operator using greater processor power. It can be seen that given more power the mutation operator improves its solution quality from average deviation of 0.731% to 0.455%. In 2005 Misevičius and Kilda studied the recombination operators of hybrid algorithms [20] and evaluated the performance of a hybrid algorithm of crossovers together with a tabu search. Comparing the results they obtained with our study brings more comprehensive conclusions. Hybrid algorithms provide much better results than pure GAs however, it is not easy to evaluate the performance of a crossover when it is implemented with a local search tool. From this perspective, the work of Misevičius and Kilda evaluated the crossovers as a diversification tool, whereas we evaluated them as intensification tools. Although the test environments and instances differ, Misevičius and Kilda determined a resulting ranking of SPX, COHX, and MPX, which is very similar to our ranking. They also indicated the outstanding performance of the less disruptive operators as we concluded from our study. The mutation is a less disruptive operator that was shown to work well in our study. Multiple parent operators such as MPX did not perform well for our problem instances whereas, they are reported to work well with hybrid algorithms in general. Although the OBX is not reported as a good operator with hybrid algorithms, it gave good results in our study.

Table 3. Long run test results for a mutation operator with 96 processors and 1,000 seconds (96,000 CPU seconds).

Instance	*Devation*	*Instance*	*Devation*	*Instance*	*Devation*	*Instance*	*Devation*
sko42	***0.290***	*tai20a*	***0.470***	*esc32a*	***0.000***	*nug20*	***0.00***
sko49	***0.310***	*tai25b*	***1.240***	*esc32b*	***0.000***	*nug21*	***0.00***
sko56	***0.150***	*tai30a*	***0.000***	*esc32c*	***0.000***	*nug22*	***0.00***
sko64	***0.240***	*tai35a*	***1.040***	*esc32d*	***0.000***	*nug24*	***0.00***
sko72	***0.190***	*tai40a*	***1.590***	*esc32e*	***0.000***	*nug25*	***0.00***
sko81	***0.470***	*tai50a*	***1.530***	*esc32f*	***0.000***	*nug27*	***0.00***
sko90	***0.600***	*tai60a*	***1.950***	*esc32g*	***0.000***	*nug30*	***0.00***
sko100a	***0.590***	*tai80a*	***1.970***	*esc32h*	***0.000***		
sko100b	***0.440***	*tai100a*	***2.340***	*esc64a*	***0.000***		
sko100c	***0.670***			*esc128*	***0.000***		
sko100d	***0.670***						

sko100e	***0.720***			
sko100f	***0.810***			
Average	***0.472***	***1.347***	***0.000***	***0.00***

Overall Average is : 0.455 %.

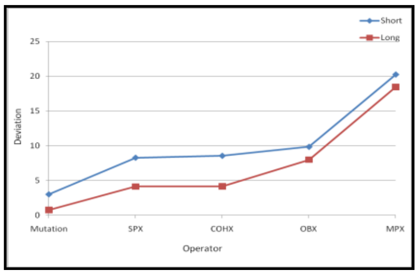

Fig. 4. Ranking of operators according to the average deviation of short and long tests from the best known results in *QAPLIB*.

5 Conclusions

In this paper, the parallel optimization of the QAP with the mutation operator was discussed. All the operators were tested in a framework of island parallel genetic algorithm with short and long runs, and a ranking of operators was determined. The experimental environment was a parallel genetic algorithm, thus it was possible to evaluate the operators clearly and in a more stabilized experimental condition. Within all the experimental conditions, the mutation operator outperformed the other selected operators. The low degree disruptive and swap based nature of the mutation operator is observed to be the best performing operator for the QAP. The mutation is a very promising operator and shows an outstanding performance compared with all the other operators. For the future work, it could be of interest to analyze how the behavior of the mutation operator improves when used with parallel hybrid genetic algorithms.

References

1. Garey, M.R., Johnson, D.S.:Computers and intractability: A guide to the theory of NP-Completeness. Freeman: San Francisco, (1979)
2. Koopmans, T.C., Beckmann, M.J.:Assignment problems and the location of economic activities. Econometrica, 25, pp.53-76, (1957)
3. Steinberg, L.:The backboard wiring problem: A placement algorithm. SIAM Review, vol. 3, pp. 37–50, (1961)
4. Rossin, D.F., Springer, M.C.,Klein, B.D.:New complexity measures for the facility layout problem: An empirical study using traditional and neural network analysis. Computers & Industrial Engineering, vol. 36, pp. 585–602, (1999)
5. Pfister, G.F.:In Search of Clusters (2nd Edition). Prentice Hall PTR, (1998)

6. Lim, M. H., Yuan, Y., Omatu, S.:Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem. Computational Optimization and Applications, vol. 15, no. 3, pp. 249–268, (2000)
7. Gelenbe, E., Timotheou, S. Nicholson, D.: Fast Distributed Near-Optimum Assignment of Assets to Tasks. Computer Journal,Vol:53, pp.1360-1369, (2010)
8. Bozdogan, A.O., Efe M.:Improved assignment with ant colony optimization for multi-target tracking Expert Systems with Applications 38, pp. 9172–9178, (2011)
9. Adl, R.K., Rankoohi, S.M.T.R.:A New Ant Colony Optimization Based Algorithm for Data Allocation Problem in Distributed Databases. Knowledge Information Systems, pp.349-372, (2009)
10. Haghani, A., Chen, M.:Optimizing gate assignments at airport terminals. Transportation Research A 32 (6), pp. 437–454, (1998)
11. Ciriani, V., Pisanti, N., Bernasconi, A.:Room allocation: A polynomial subcase of the quadratic assignment problem. Discrete Applied Mathematics, 144, 263–269, (2004)
12. Cordeau, J-F.,Gaudioso, M., Laporte, G., Moccia, L.:The service allocation problem at the GioiaTauro Maritime Terminal. European Journal of Operational Research, Forthcoming, (2005)
13. Lawler, E.L.:The Quadratic Assignment Problem. Management Science, 9, pp. 586-599., (1963)
14. Holland, J.H.:Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, USA, (1975)
15. Goldberg, D.:Genetic algorithms in search, optimization & machine learning. Addison-Wesley, Reading, Mass, (1989)
16. Sevinc, E., Cosar, A.:An Evolutionary Genetic Algorithm for Optimization of Distributed Database Queries. The Computer Journal, vol.54, issue: 5,pp. 717-725, (2011)
17. MiseviciusA.,BronislovasK.:Comparison of crossover operators for the quadratic assignment problem. Information Technology and Control, Vol.34, No.2, (2005)
18. Cantu-Paz.:Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, (2000)
19. Tosun,U., Dokeroglu, T., Cosar, A., (submitted).:A New Robust Island Parallel Genetic Algorithm for the Quadratic Assignment Problem. Expert Systems with Applications, 2012.
20. Misevicius, A., Rubliauskas, D.:Performance of hybrid genetic algorithm for the grey pattern problem. Information Technology and Control, Vol.34, No.1, 15−24, (2005)
21. Ahuja, R.K., Orlin,J.B.,Tiwari,A.:A greedy genetic algorithm for the quadratic assignment problem. Computers & Operations Research, Vol.27, 917–934,(2000)
22. Drezner, Z.:A new genetic algorithm for the Quadratic Assignment Problem. INFORMS J. Comput., 15, pp. 320-330, (2003)
23. Lim, D., Ong, Y.-S., Jin, Y., Sendhoff, B., Lee, B.-S.:Efficient hierarchical parallel genetic algorithms using grid computing. Future Gener. Comput. Syst., 23(4), pp.658–670,(2007)
24. Tsutsui, S., Fujimoto, N.:Solving Quadratic Assignment Problems by Genetic Algorithms with GPU Computation: a Case Study. GECCO, Montreal Quebec, Canada, (2009)
25. Vázquez, M., Whitley, L.D.: A Hybrid Genetic Algorithm for the Quadratic Assignment Problem. In GECCO, 135-142, (2000)
26. Burkard, R. E., Karisch, S. E.,Rendl, F.:QAPLIB a quadratic assignment problem library. *Journal of Global Optimization*, 10, pp.391–403, (1997)