New Benchmark Instances for the QAP and the Experimental Analysis of Algorithms

Thomas Stützle[1] and Susana Fernandes[2]

[1] Darmstadt University of Technology, Computer Science Department, stuetzle@informatik.tu-darmstadt.de

[2] Universidade do Algarve, Departamento de Matemática, sfer@ualg.pt

Abstract. The quadratic assignment problem arises in a variety of practical settings. It is known to be among the hardest combinatorial problems for exact algorithms. Therefore, a large number of heuristic approaches have been proposed for its solution. In this article we introduce a new, large set of QAP instances that is intended to allow the systematic study of the performance of metaheuristics in dependence of QAP instance characteristics. Additionally, we give computational results with several high performing algorithms known from literature and give exemplary results on the influence of instance characteristics on the performance of these algorithms.

1 Introduction

The QAP can best be described as the problem of assigning a set of objects to a set of locations with given distances between the locations and given flows between the objects. The goal is to place the objects on locations in such a way that the sum of the product between flows and distances is minimal. It is a model of many real world problems arising in hospital layout, keyboard layout and other areas [4,5].

More formally, given n objects and n locations, two $n \times n$ matrices $A = [a_{ij}]$ and $B = [b_{rs}]$, where a_{ij} is the distance between locations i and j and b_{rs} is the flow between objects r and s, the QAP can be stated as

$$\min_{\phi \in \Phi} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{\phi_i \phi_j} b_{ij} \qquad (1)$$

where Φ is the set of all permutations of the set of integers $\{1, \ldots, n\}$, and ϕ_i gives the location of object i in the current solution $\phi \in \Phi$.

The QAP is a $\mathcal{NP}$-hard optimization problem [17]. It is considered as one of the hardest optimization problems since the largest instances that can be solved today with exact algorithms are limited to instances of size around 30 [1,13]: the largest, non-trivial instance from QAPLIB [12], a benchmark collection for the QAP, solved to optimality has 36 locations [3]. In practice, the only feasible way to solve large QAP instances is to apply heuristic algorithms which find very high quality solutions in short computation time. Several such algorithms have been proposed which include algorithms like simulated annealing [6], tabu search [2,18,22], genetic algorithms [10,16,8], GRASP [14], ant algorithms [11, 15,21,20], and scatter search [7].

Previous research results show that the characteristics of the QAP instances strongly influence the relative performance of the various algorithmic techniques for the QAP [23,11,21,19]. Based on the input data, two measures were identified that have significant influence on the relative performance of metaheuristics, the *(flow) dominance* that corresponds to the variation coefficient, i.e. the standard deviation of the (flow) matrix entries divided by their average value, and the *sparsity* of the matrix, i.e. the fraction of zero entries [16,21]. Similarly, the performance of metaheuristics for the QAP has been related to search space characteristics like ruggedness and fitness distance correlations [16,21]. However, currently there is still a strong lack of (i) a complete understanding of how exactly the relative performance of different metaheuristics depends on instance characteristics, (ii) knowledge of how input data characteristics of QAP instances and search space features relate one to each other, and (iii) even the relative performance of algorithms is not clear because of the different hardware and implementation details used in the various researches. Our goal is to improve this situation through the introduction of a new, large set of benchmark instances that offers a range of instances with different sparsities and flow dominances and using these instances for a systematic analysis of metaheuristic implementations that are as much as possible based on the same data structures. In this paper, Section 2 describes the set of new benchmark instances and Section 3 exemplifies the analysis of the influence of QAP instance characteristics on the relative performance of various metaheuristic implementations. We conclude in Section 4.

2 Benchmark Instances

2.1 Instances from QAPLIB

It is known that the particular type of a QAP instance has a considerable influence on the performance of heuristic methods [23]. For example, the instances available from QAPLIB have been classified by Taillard into four classes. Despite the fact that QAPLIB comprises about 130 instances, there are several disadvantages associated for experimental analysis. First, many of the QAPLIB instances are too small to pose a real challenge to the best available metaheuristic algorithms for the QAP and only a small number of instances of a size with $n \geq 50$ remain (the largest instance is of size $n = 256$ and only one instance of this size exists).[1] Second, with a small number of instances, one may introduce a strong

[1] For example, the high performing iterated local search algorithm from Stützle [19] on all except 18 instances from QAPLIB finds the proven optimal or best known solutions in every single trial of limited length (these results were measured across at least 25 trials of a maximal length of 1200 seconds on a 1.2 GHz Athlon processor); most of these instances are actually "solved" on average within few seconds. However, most QAPLIB instances, including also the small ones of size about 20, are still very challenging for exact algorithms.

bias when solving these instances with metaheuristics due to overfitting effects. Third, the instances in QAPLIB do not vary systematically in their characteristics and therefore their utility is very limited for analyzing the performance of metaheuristics in dependence of instance characteristics.

Other, new instances that were designed to be hard were proposed by Taillard and Drezner [9]. Taillard's instances are available at `http://ina.eivd.ch/collaborateurs/etd/problemes.dir/qap.dir/qap.html` and Drezner's instances are available at `http://business.fullerton.edu/zdrezner/programs.htm`. However, these instances have very particular features and the instance characteristics of the two new classes of instances are rather similar, limiting the scope of experimental studies. Nevertheless, these instances are interesting for studies that examine, for which type of instances metaheuristic instances may fail.

2.2 New Random QAP Instances

Because of the limitations associated with the instances from QAPLIB and those of other sources, we generated a large number of instances with varying characteristics. The set of instances was generated in such a way that (i) instance characteristics are systematically varied and (ii) that it is large enough to allow systematic studies on the dependence of the performance of metaheuristics on instance characteristics. These instances were also used in the experimental evaluation of several metaheuristics in the Metaheuristics Network (see also `www.metaheuristics.net`) and are available at URL `www.intellektik.informatik.tu-darmstadt.de/~tom/qap.html`.

Instance generation. The instance generation is done in such a way that instances with widely different syntactic features concerning the flow dominance values of the matrix entries (the dominance value is the variation coefficient of the matrix entries) and the sparsity of the matrix entries (which measures the percentage of zero matrix entries) can be generated. These two measures strongly influence the characteristics of the QAP instances and are conjectured to also have a strong influence on metaheuristics' performance. As a second important parameter we considered two different ways of generating the distance matrix, which allows to generate distance matrices with different structures. In particular, for the *distance matrix* we applied two different ways of generating the distance matrix.

Euclidean distances: In a first approach the distance matrix corresponds to the Euclidean distance between n points in the plane. These points are generated in such a way that they fall into several clusters of varying size. The characteristics of the cluster (location, number of points) are generated as follows

- generate a random number k according to a uniform distribution in $[0, K]$, where K is a parameter.
- generate a cluster center (c_x, c_y), where c_x and c_y are randomly chosen according to a uniform distribution in $[0, 300]$.

- randomly choose the coordinates of the k cluster points in a square centered around (c_x, c_y):
  1. generate a and b randomly according to a uniform distribution in $[-m/2, m/2]$, where m is a parameter,
  2. set $x = c_x + a$ and $y = c_y + b$.

These steps are repeated until n points have been generated.

Manhattan distances: In a second approach, the distance matrix corresponds to the pairwise distances between all the nodes on a $A \times B$ grid, where the distance is calculated as the Manhattan distance between the points on a grid.

The reasons for the choice of the different distance matrices are the following. Instances based on Euclidean distances will have most probably only one single optimal solution and by different parameter settings for the generation of the distance matrices, the clustering of locations often encountered in real-life QAP instances can be imitated. Instances with a distance matrix based on the Manhattan distance of points on a grid have, due to symmetries in the matrix, at least four optimal solutions which are at a maximum distance possible from each other[2]. Moreover, the distance matrices show a much lower clustering. Additionally, some real-life instances have distances derived from points on a grid.

Different ways of generating the *flow matrix* are also considered. The flow matrix for all instances are asymmetric and have a null diagonal. Here, in particular, we generate instances which have a strongly varying flow dominance and sparsity. For the generation of the flow matrix we considered two different cases: (i) the flow entries are randomly generated, (ii) flow entries are structured in the sense that clusters of objects exist which have a significant interaction.

Random flows: The generation of the random flow matrices uses the following parameters:
- sp, with $0 \leq sp < 1$, indicates the sparsity of the flow matrix
- a and b determine the flow values.

Each flow matrix entry is then generated by the following routine

```
double x = random->next(); % a random number in [0,1]
if ( x < sp )
  return 0;
else {
  x = random->next();
  return (unsigned long int) MAX(1,pow((a * x),b));
}
```

where x is a random number uniformly distributed in $[0, 1]$. By varying sp we can generate instances of arbitrary sparsity. The parameters a and b allow to vary the distribution of the matrix entries according to $y = (a \cdot x)^b$.

[2] As the distance between two solutions we use here a straightforward extension of the Hamming distance that counts the number of different positions in two permutations corresponding to the number of different assignments.

The instances below are generated with the following combinations of a and b: $(100, 1)$, which corresponds to uniformly distributed matrix entries in $[0, 100]$ (the same as the unstructured instances of one of the instances classes defined by Taillard [23]), $(10, 2), (3.5, 3), (2, 7)$. For fixed value of sp, the instances with larger exponent tend to have higher flow dominance.

Structured flows: In real applications, groups of objects tend to have typically a large interaction, while the interaction among different groups tends to be smaller. Instances with structured flow matrix entries take these properties into account. This is done in an indirect way as follows: We generate n points (objects) randomly according to a uniform distribution in a square of dimension 100×100. In the flow matrix a flow between two objects i and j exists if the distance between the points i and j is below some constant threshold value d: "Close" objects tend to have flow, while for "far" objects the flow is zero. The non-zero flow entries are generated like in the random flow case as $\max\{1, (a \cdot x)^b\}$. Note that the threshold value d has a strong influence on the sparsity (the smaller d, the less objects will exist which exchange flow) and the flow dominance of the flow matrix.

Structured flows plus: In the flow generation of the structured flows, distant objects will not exchange any flow. In a straightforward extension, we generate with a small probability p flows between objects for which the associated points have a distance larger than the threshold d. The non-zero flow entries are generated like in the random flow case as $\max\{1, (a \cdot x)^b\}$.

Benchmark instances. We distinguish among six different classes of instances that differ in the way different variants for the distance matrix and the flow matrix are combined. The class identifiers are

RandomRandom: Random distance matrix and random flows;
RandomStructured: Random distance matrix and structured flows;
RandomStructuredPlus: Random distance matrix and structured flows with connections among clusters of objects;
GridRandom: Grid-based distance matrix and random flows;
GridStructured: Grid-based distance matrix and structured flows;
GridStructuredPlus: Grid-based distance matrix and structured flows with connections among clusters of objects.

For each of the six classes we have randomly generated instances of different sizes with $n \in \{50, 100, 150, 200, 300, 500\}$. For each class a number of instances differing in the above defined parameters for generating the matrix entries were generated resulting in a total of 644 new benchmark instances.

To give an impression of the variety of the instance characteristics within these instances, in Figure 1 we indicate the variation of the flow dominance and the sparsity of the instances across all the 136 instances of size 50. In that plot four different curves are visible. This pattern is due to the influence of the parameters a and b on the generation of the flow matrix. In general, the plot shows that there is a strong, non-linear relationship between the flow dominance and

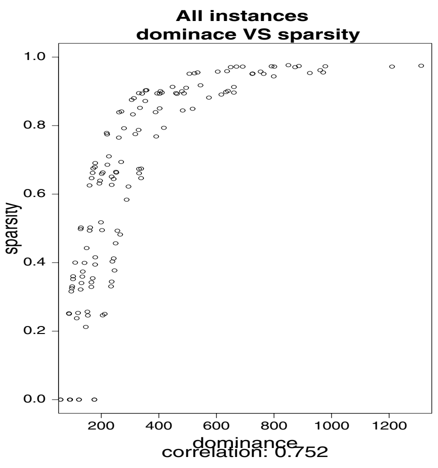

Fig. 1. Plots of the relationship between flow dominance and sparsity of the instances of size 50. On the x-axis is given the flow dominance and on the y-axis is given the sparsity of the instances.

the sparsity. Similarly, we investigated properties like autocorrelation length and the number of improvement steps for reaching a local optimum; these measures give some hint on the ruggedness of the search landscape.

3 Experimental Results

As said, the set of new instances is intended to allow the study of the performance of metaheuristics in dependence of instance characteristics. In this section we give an example of the experimental results and insights that can be obtained using the set of new instances. Here, we limit our analysis to instances of size 50 and give performance results for six different algorithms. In particular, we run robust tabu search (RoTS) [22], a simulated annealing (SA) algorithm [6], $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [21], two iterated local search (ILS) variants, a population-based one (ES-ILS) and a non-population based one (ILS) [19], and fast ant system (FANT) [24]. All the algorithms were using the parameter settings as proposed by their original authors. The choice of these algorithms was based on the desire to have different types of metaheuristics known to perform well for specific classes of instances. (In an extended version of this article we will include a number of additional algorithms including most of the current well known state-of-the-art algorithms.)

All these algorithms were implemented in C based on the same underlying local search procedure and using the same data structures as much as possible. Each algorithm was run 10 times on each of the instances for 150 seconds on a Pentium III 700 MHz processor under SUSE Linux 7.1. Note that the stopping

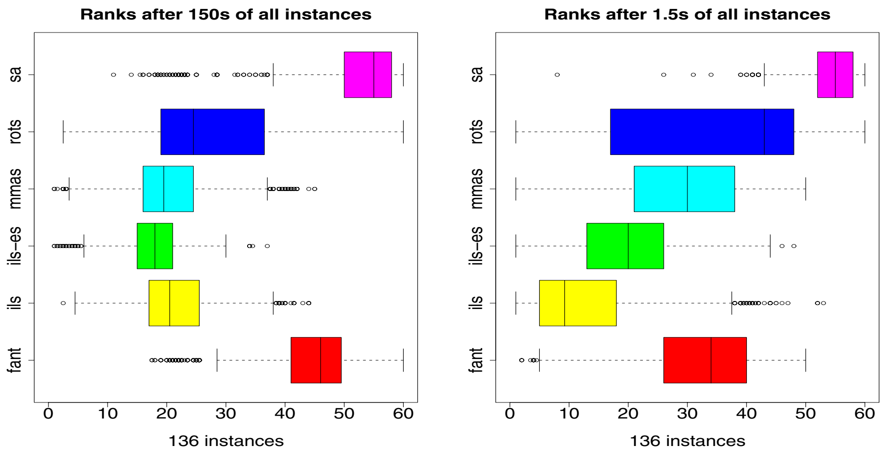

Fig. 2. Boxplots of the ranks for the six metaheuristics measured across all the instances of size 50 (left: stopping criterion 150 seconds; right: stopping criterion 1.5 seconds). The x-axis gives the rank for each single solution found by an algorithm. Note that for each instance a total of 60 trials was run and therefore on each instance the range of the ranks is from 1 to 60.

criterion of 150 seconds was chosen in such a way that RoTS can do about $10\,000n$ iterations on our computer, where n is the instance size.

The experimental analysis was based on ranking the results returned by the algorithms. For every algorithm the best solution achieved in each single trial was saved and then all these values were ranked. Since each of the six algorithms was run ten times, this results overall in 60 possible ranks. Based on the ranks achieved by each algorithm, we used boxplots to visualize the results achieved. In the boxplots, the central box shows the data between the two quartiles, with the median being represented by a line. "Whiskers" extend to the most extreme data point which is no more than 1.5 times the interquartile range from the box. If there are data points further away than the extensions of the whiskers, these data are indicated by small circles.

In Figure 2 we give a summary of the results when ranking across all the 136 instances of size 50 (hence, each box is based on 1360 data points), when stopping the algorithms after 150 seconds (left side) and when stopping the algorithms after 1.5 seconds (right side). The plots suggest that there are significant differences among the algorithms, which is confirmed by using Friedman rank sum tests that rejects the null hypothesis that says that all the results are equal. (We applied the Friedman rank sum test to all the data presented in the following figures; in all cases this test indicates that there exist significant differences in the performance of the algorithms.) In fact, for the largest computation times, ES-ILS appears to be the best performing algorithm followed by $\mathcal{MM}$AS and ILS. The worst performing algorithms are FANT and SA. The ranking of the algorithms is different, however, if only short computation times are allowed.

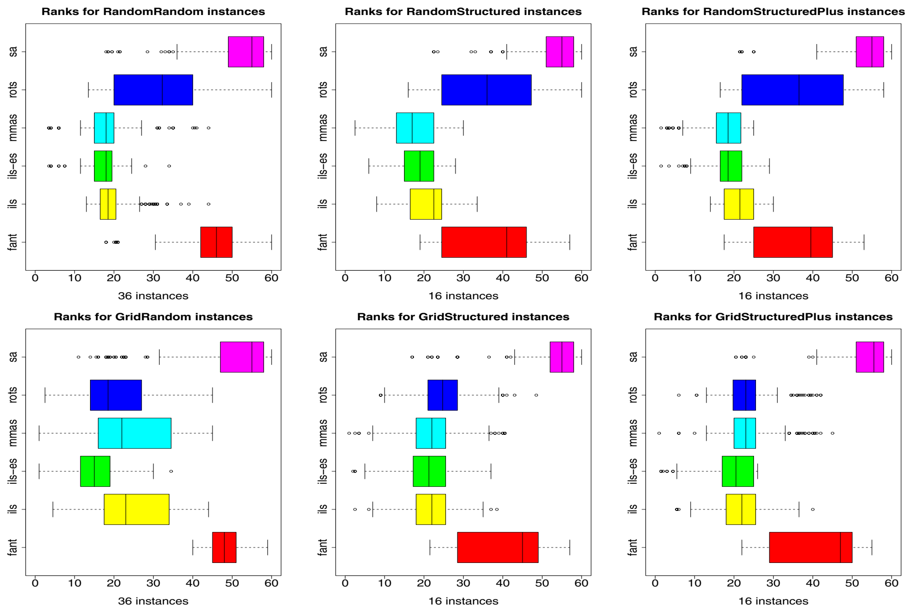

Fig. 3. Boxplots of the ranks for the six metaheuristics measured across the six different instance classes. The x-axis gives the rank for each single solution found by an algorithm. The upper plots are random distance matrices, while the lower plots are for instance classes where the distance matrix is based on a grid.

In fact, when allowing only 1.5 seconds of computation time, ILS gives the best results followed by ES-ILS, $\mathcal{MM}$AS, and RoTS and FANT. The reason that ILS is now much better performing than either ES-ILS or $\mathcal{MM}$AS is that the latter two algorithms are population-based algorithms and with short computation times, they were only able to run few iterations. Differently, ILS uses only one single solution at each step and therefore it is also quicker in identifying high quality solutions early in the search. It is also noteworthy that FANT is now, relative to the other algorithms, much better performing than for longer trials. In fact, FANT was designed to achieve good solutions quickly [24].

Further analysis was done to investigate the dependence of the relative performance of the algorithms on the type of instances and on features like the flow dominance. In Figure 3 we give results for the six different instance classes for the 150 seconds time limit (see page 203 for a description of the six classes). The results show that the class of instances can have significant influence of the relative performance of the metaheuristics tested here. For example, on the instances with random distance matrix, FANT catches up in performance with RoTS, while it is significantly worse than RoTS on instances with Grid distance matrix. Similarly, $\mathcal{MM}$AS catches up with ES-ILS on the instances with random distance matrix, while on the instances with grid distance matrix ES-ILS appears to be superior to $\mathcal{MM}$AS.

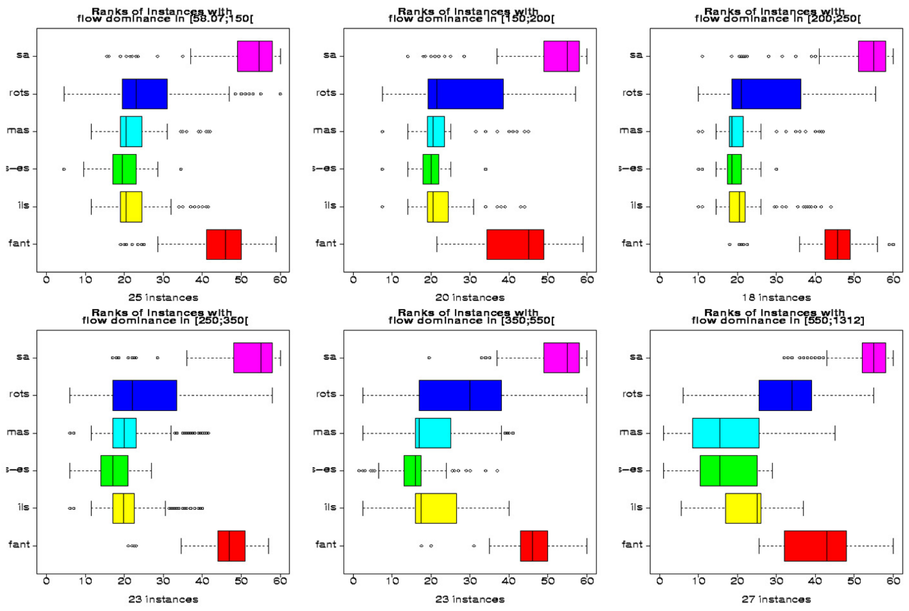

Fig. 4. Boxplots of the ranks for the six metaheuristics measured across the instances classified by the flow dominance. The x-axis gives the rank for each single solution found by an algorithm. Note that for each instance a total of 60 trials was run and therefore on each instance the range of the ranks is from 1 to 60.

In Figure 4 we present the results in dependence of the flow dominance values of the various instances. In particular, we divided the range of observed flow dominance into six intervals from $[58; 150]$, $]150; 200]$, $]200; 250]$, $]250; 350]$, $]350; 550]$, $]550; 1312]$. For increasing flow dominance, the performance of RoTS is decreasing when compared to ES-ILS, ILS, and $\mathcal{MM}$AS, which can be observed, for example, in the increasing differences concerning the medians obtained by the various algorithms. When comparing ES-ILS to $\mathcal{MM}$AS it becomes clear that the advantage of ES-ILS over $\mathcal{MM}$AS is mainly because of the better performance on instance with medium values of the flow dominance in the range from 250 to 550. This observations is true for the largest time limits, while for shorter time limits the advantage of ES-ILS over $\mathcal{MM}$AS appears to be larger (the corresponding plots are not shown here); this may suggest that $\mathcal{MM}$AS is taking longer time to converge than ES-ILS.

4 Conclusions

In this paper we introduced a new set of QAP instances that is intended for the systematic analysis of the performance of metaheuristics in dependence of instance characteristics. We exemplified the use of these classes exemplifying the performance differences of six well known metaheuristic implementations using an analysis based on ranking procedures. Currently we are extending the scope of this analysis strongly by (i) including a larger number of metaheuristics into the performance analysis, (ii) applying the algorithms also to the larger instances and to all QAPLIB instances, and (iii) examining the dependence of the algorithms' performance on measures based on search space characteristics (instead of pure syntactic features of the instance date) like fitness-distance correlation, the ruggedness of the search spaces, or the occurrence of plateaus in the search space. The ultimate goal of this research is to deepen our understanding of the performance of different metaheuristics on the QAP and, more in general, to learn about the relation between metaheuristic performance and search space characteristics of the problem under solution.

Acknowledgments. We would like to thank Luis Paquete for his suggestions on this research and his help with R and several scripts to extract the results. Susana acknowledges support from a joint DAAD-ICCTI project. This work was supported by the Metaheuristics Network, a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

1. K. M. Anstreicher, N. W. Brixius, J.-P. Goux, and J. Linderoth. Solving large quadratic assignment problems on computational grids. *Mathematical Programming*, 91(3):563–588, 2002.
2. R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.
3. N. W. Brixius and K. M. Anstreicher. The Steinberg wiring problem. Technical report, College of Business Administration, University of Iowa, Iowa City, USA, October 2001.
4. R. E. Burkard, E. Çela, P. M. Pardalos, and L. S. Pitsoulis. The quadratic assignment problem. In P. M. Pardalos and D.-Z. Du, editors, *Handbook of Combinatorial Optimization*, volume 2, pages 241–338. Kluwer Academic Publishers, 1998.
5. E. Çela. *The Quadratic Assignment Problem: Theory and Algorithms.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.
6. D. T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46(1):93–100, 1990.
7. V.-D. Cung, T. Mautor, P. Michelon, and A. Tavares. A scatter search based approach for the quadratic assignment problem. In T. Baeck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 165–170. IEEE Press, Piscataway, NJ, USA, 1997.
8. Z. Drezner. A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, 15(3):320–330, 2003.

9. Z. Drezner, P. Hahn, and É. D. Taillard. A study of quadratic assignment problem instances that are difficult for meta-heuristic methods. *Annals of Operations Research*, to appear.
10. C. Fleurent and J. A. Ferland. Genetic hybrids for the quadratic assignment problem. In P. M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 173–187. American Mathematical Society, Providence, RI, USA, 1994.
11. L. M. Gambardella, É. D. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50(2):167–176, 1999.
12. P. Hahn. QAPLIB - a quadratic assignment problem library. `http://www.seas.upenn.edu/qaplib/`, 2003. Version visited last on 15 September 2003.
13. P. M. Hahn, W. L. Hightower, T. A. Johnson, M. Guignard-Spielberg, and C. Roucairol. Tree elaboration strategies in branch and bound algorithms for solving the quadratic assignment problem. *Yugoslavian Journal of Operational Research*, 11(1):41–60, 2001.
14. Y. Li, P. M. Pardalos, and M. G. C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P. M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, Providence, RI, USA, 1994.
15. V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.
16. P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4):337–352, 2000.
17. S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.
18. J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2(1):33–45, 1990.
19. T. Stützle. Iterated local search for the quadratic assignment problem. Technical Report AIDA-99-03, FG Intellektik, FB Informatik, TU Darmstadt, 1999.
20. T. Stützle and M. Dorigo. ACO algorithms for the quadratic assignment problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 33–50. McGraw Hill, London, UK, 1999.
21. T. Stützle and H. H. Hoos. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.
22. É. D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4–5):443–455, 1991.
23. É. D. Taillard. Comparison of iterative searches for the quadratic assignment problem. *Location Science*, 3(2):87–105, 1995.
24. É. D. Taillard. FANT: Fast ant system. Technical Report IDSIA-46-98, IDSIA, Lugano, Swiss, 1998.