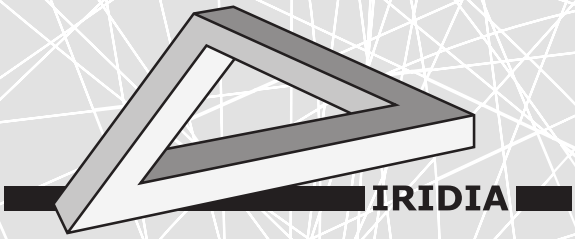

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Tabu Search vs. Simulated Annealing for Solving Large Quadratic Assignment Instances

Mohamed Saifullah Hussin, Thomas Stützle

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2010-020

October 2010

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2010-020

Tabu Search vs. Simulated Annealing for Solving Large Quadratic Assignment Instances

Mohamed Saifullah Hussin and Thomas Stützle

October 2010

Abstract

Performance comparisons between algorithms have a long tradition in metaheuristic research. An early example are comparisons between Tabu Search (TS) and Simulated Annealing (SA) algorithms for tackling the Quadratic Assignment Problem (QAP). The results of these comparisons are to a certain extent inconclusive, even when focusing on only these two types of algorithms. While comparisons of SA and TS algorithms were based on rather small-sized instances, here we focus on possible dependencies of the relative performance between SA and TS algorithms on instance size. In fact, our experimental results show that the assertion whether one algorithm is better than the other can depend strongly on QAP instance size even if one focuses on instances with otherwise same charateristics.

1 Introduction

The Quadratic Assignment Problem (QAP) has attracted an enormous amount of research efforts [5, 6, 10] so that nowadays it is one of the most widely studied NP-hard combinatorial optimization problems. The QAP is often used to give an abstract formulation of real world applications arising in the layout of hospitals, factories, or keyboards. In the QAP are given two $n \times n$ matrices A and B, where a_{ij} gives the distance between the pair of positions i and j and b_{kl} gives the amount of flow exchanged between units k and l. Assigning units k and l to positions i and j, respectively, incurs in a cost contribution of $a_{ij} \cdot b_{kl}$. The goal of the QAP is to assign units to positions such that the sum of all cost contributions is minimized. A candidate solution for the QAP can be represented as a permutation π, where $\pi(i)$ is the unit that is assigned to position i. The objective function of the QAP can then be given as

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot b_{\pi(i)\pi(j)}. \tag{1}$$

The QAP is well known for its hardness. Until today, the largest, non-trivial instance solved to proven optimality from QAPLIB (http://www.seas.upenn.edu/qaplib/), a benchmark library for the QAP, is of dimension 36. Researchers therefore opted for the use of metaheuristics to effectively solve the

QAP. Among the first metaheuristics that have been applied to the QAP are Tabu Search (TS) and Simulated Annealing (SA). In fact, for most QAPLIB instances the nowadays best known solutions have been found first by either TS or SA algorithms, demonstrating in this sense the effectiveness of these two algorithms.

Previous research findings on the performance comparison between TS and SA suggest that TS performs better than SA [1, 3, 14]. Battiti and Tecchiolli [3] reported that TS performed better than SA in terms of CPU time needed to reach a solution quality which is 1% from the best known solutions. The instances considered, however, are rather small. Another direct comparison between SA and TS was done by Chiang and Chiang [7] where they have compared the performance of SA, TS, Probabilistic TS, and Hybrid TS for solving the facility layout problem, formulated as a QAP. Their results show that their basic TS approach performs better than SA. Other researchers have compared the performance of SA and TS with other algorithms. The earliest one is the comparison of SA, TS, Genetic Algorithms (GA), Great Deluge Algorithm (GDA), and Record-to-Record Travel (RRT) for solving the QAP that were conducted by Sinclair [14]. His results on the comparisons between TS and SA show that TS provided better solutions than SA in 28 out of 37 cases. Comparisons between SA and TS with a hybrid ant colony system with local search, HAS-QAP were reported in [9]. The results suggest that the performance of TS is better than SA on most of the instances tested. A more recent article by Arostegui et al. [1] reported the comparison between SA, TS and GA on a specific QAP domain, the facilities location problems (FLP). They compared the performance of these algorithms on several variants of FLP. When comparing TS and SA, their results show that TS surpasses SA in most instances.

In contrast to these results, in an early study Paulli [13] has reported that SA outperforms TS when the same computation time is spent by both algorithms. While this result may be attributed to the implementation of TS without the speed-ups proposed by Taillard [16], in a very recent study, Paul [12] shows that when using an instance-dependent restart strategy, an SA algorithm may perform actually better than TS algorithms when high solution quality is required. However, it is unclear how this result generalizes to general, instance-independent restart mechanisms.

In this study, we examine how instance size influences the relative performance of TS and SA algorithms for solving the QAP. We have considered a range of QAP instance sizes from 20 up to 500 items; in fact, the large instances we use are way beyond the typical instance sizes tackled in most previous studies. Initially, we conducted our study on randomly generated instances that show a structure that is considered to be usual for real-life type QAP instances [17]. Given the observation of clear transitions in the relative performance of SA and TS algorithms, we extended the study also to other instance classes. For the experimental tests, we considered two TS variants and two SA variants, which are shortly presented in the next section.

This article is structured as follows. In the next section, we first introduce the algorithms considered in this study, followed by introduction on benchmark instances we use in this article. Experimental results with the comparison between

Algorithm 1 Outline of the SAR algorithm

procedure *Simulated Annealing-Restart*
generate initial solution π
$\pi_{best} = \pi$
initial value for T_0
while termination criterion not satisfied **do**
  $\pi' = GenerateRandomNeighbor(\pi)$
  $\pi'' = Accept(T_n, \pi, \pi')$
  if $f(\pi'') < f(\pi_{best})$ **then**
    $\pi_{best} = \pi''$
  end if
  T_{n+1}=$SetTemperature(T_n, Iterations, c)$
  $\pi = \pi''$
end while
return π_{best}
end *Simulated Annealing*

RoTS, RTS, SAC, and SAR are given in Section 4 and we conclude in Section 5.

2 Algorithms

In this article, we compare the performance of two TS variants and two SA variants for the QAP. For TS, we have re-implemented the RoTS by Taillard [16], which was reported to achieve very good performance on the QAP, and we use the RTS code by Battiti and Tecchiolli [2] to which we included some minor changes to have a faster function evaluation matching the performance of our re-implementation of RoTS. Both algorithms are run using their default parameter settings suggested in earlier articles.

As SA algorithms, we re-implemented the SA scheme Q8-7 of Connolly [8] (called here SAC for SA Connolly), which was the recommended scheme by Connolly, and we implemented a basic SA algorithm we call SAR. This algorithm is described next.

Our SAR algorithm is essentially a standard SA algorithm that uses a temperature reheating during the run. The temperature reheating is intended to allow the algorithm to escape from particular local minima regions when the temperature reaches a very low value. An algorithmic outline of SAR is given in Algorithm 1. Another restart-based simulated annealing was proposed earlier by Wang [18] for solving the QAP.

SAR starts by selecting an initial permutation uniformly at random. This permutation is set as the current best solution π_{best} and current best evaluation is set as $f(\pi_{best})$. The initial temperature is set to $T_0 = 0.005 \cdot f(\pi_{best})$, which resulted in reasonable performance. In function *GenerateRandomNeighbor(π)*, a neighbor π' of π is obtained by swapping the position of two items i and j. We apply the pair-swops sequentially for each solution π for all $k = \{1, \ldots, n\}$,

following the suggestion of Connolly [8]. All moves that improve the current candidate solution are accepted, while moves that worsen it are accepted based on the Metropolis condition. This acceptance criterion is implemented by the function $Accept(T_n, \pi, \pi')$. For our cooling schedule, we retain the temperature level for $c \cdot n$ consecutive swaps, where c is a parameter. Some initial experiments showed that $c = 100$ results in satisfactory performance. We use geometric cooling, with the temperature at iteration $i+1$ being set to $T_{i+1} = \alpha \cdot T_i$ with $\alpha = 0.9$. When the temperature drops below 1, we reset the temperature value to T_0 and restart the same cooling process. This temperature schedule is implemented in function $SetTemperature(T_n, Iterations, c)$. The SA algorithm continues until the maximum computation time limit.

3 Benchmark Instances

Most studies of algorithms for the QAP are based on instances from QAPLIB. However, the number of large instances in QAPLIB is limited and it does not allow for a systematic study of algorithms, in particular, if the dependency of solver performance on instance size is of interest. Therefore, we have generated a new, large set of QAP instances of various sizes.

We have generated three sets of benchmark instances. The first set has Euclidean distance matrices and structured flow matrices (ES). The distance matrix entries are generated as the pairwise Euclidean distances between n points on a plane where coordinates are generated randomly according to a uniform distribution in $[0, 100]$. The Euclidean distances are rounded to the nearest integer. For the flow matrix entries, we generate n points randomly based on a uniform distribution in a square with dimension $100x100$. If the distance between two points i and j is above a threshold value p, the flow is zero. Else, the flow is calculated as $x = -1 \cdot \log(R)$. R is a random number in $[0, \cdots, 100]$, and $flow = \min\{100 \cdot x^{2.5}, 3000\}$. All entries of both distance and flow matrices are divided by two, so that the evaluation function value of the QAP can be computed by all algorithms used on the 32-bit system we used for the computations. We divide this instance set into five classes that differ in the sparsity of the flow matrices (defined as the percentage of zero entries) and, thus, also the distance or flow dominance values (that is, the variation coefficient of the distance and flow matrix entries multiplied by 100). Instances are named based on the type of distance and flow matrices and the sparsity of flow matrix, e.g., 20.ES.0.25 refers to instance size 20 with Euclidean distance matrix and structured flow matrix with a sparsity around 0.25. We have generated for each instance class (size and sparsity value) 30 instances. This results in a total of 1 500 instances.

Note that the ES instances are the most interesting, since their flow matrix entries resemble the structure (fofr example, as measured by the matrix sparsity and the flow dominance) as it is rather usually found in real-life type QAP instances [17]. After our initial results on these instances (see next section), we generated two additional sets of instances that structurally are very difference from the ES set. For the second set of instances, we have generated instances following an article by Taillard [16]. These instances have a structure similar to the taiXXa instances from QAPLIB. All their distance and flow matrix entries

are generated based on a uniform distribution in the range $[0, 100]$. The sparsity of this instance set is approximately zero. We denote this set of instances as Randon Random (RR). The third instance set comprises instances where the distance matrix is based on Manhattan distances between points on a grid. For the flow matrices, we retain the matrix entries from the RR instance set to make them very different from the ES instances. In fact, grid-based instances are frequent in QAPLIB; for example, the instances from Nugent [11] and Skorin-Kapov [15] are such. We name this new instance set as Grid-Random (GR) instances. Each of the two sets, RR and GR, has 300 instances.

All instances are available from the page `http://iridia.ulb.ac.be/supp/IridiaSupp2010-009/`, where also their cross-statistical characteristics (sparsity, flow and distance dominance) are given.

4 Experimental Results

4.1 Experimental setup

The experiments reported in this article have been run on Intel Xeon 2.4 Ghz quad-core CPUs with 6MB cache and 8 GB of RAM running under Cluster Rocks Linux. The stopping criteria for the experiments were based on the CPU time (T) taken by RoTS to perform $10000 \cdot n$ iterations, where n is the instance size. However, we examine the algorithms' performance also in dependence of other stopping criteria, in particular, after the computation times that would correspond to $100n$, $1000n$, and $10000n$ iterations of RoTS; we refer to these three levels of the computation time as short ($100n$), medium ($1000n$), and long ($10000n$). All algorithms were run once on each instance [4]. All algorithms studied are coded in `C` and compiled with `gcc` version 3.4.6. The statistical test used for verifying the significance of the differences among algorithms is the pairwise Wilcoxon test with Holm corrections for multiple comparisons, with $\alpha = 0.05$. In the following, we give an aggregate view of the computational results; for more detailed numerical results, we refer to the supplementary page available at `http://iridia.ulb.ac.be/supp/IridiaSupp2010-009/`.

4.2 Comparison of RoTS, RTS, SAC, and SAR

Our comparison initially focused on the ES instances class, which is the most interesting ones. On each instance, we rank the solution qualities reached by the four algorithms (RoTS, RTS, SAC, and SAR) for each termination criterion (*short*, *medium*, and *long* computation times). The best algorithm is given rank 1 and the worst rank 4. If the same solution quality was obtained by more than one algorithm, the average rank is given to the corresponding algorithms. For the final ranking, we compute the average rank obtained by the algorithms across all instance classes of the same size.

Figure 1 shows plots of the average rank of algorithms when solving ES instances. There are clear dependencies of the relative rankings of the algorithms on the instance size but also on the computation time limit considered. For example,

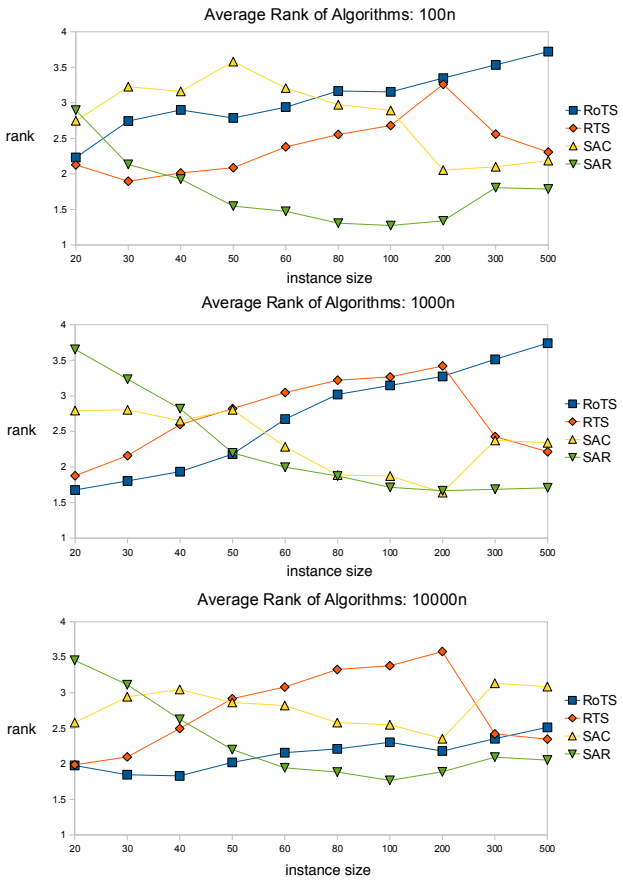

Figure 1: Line plots showing the average rank for RoTS, RTS, SAC, and SAR for ES instances at $100n(short)$, $1000n(medium)$, and $10000n(long)$.

SAR is the worst or second worst ranked algorithm on small instance sizes (20 to 40, depending on the computation time limit) but the best ranked one for instance sizes beyond 60 for all computation time limits. Similarly, RoTS has the worst rank when the instance size is 80 and above for *short* computation times. For larger computation time limits this trend, the worsening of the ranking with increasing instance size, persists, but the overall ranking of RoTS with respect to the other algorithms improves. RTS shows a very particular pattern: it is the best or second best ranked algortihm for the smallest instances and for the largest ones, but the worst or second worst ranked algorithm for intermediate sizes (in the range of 50 to 200). SAC has the worst rank on small instances at short computation times. However, its ranking is improving for larger instance sizes (at small or medium computation time limits) and for various instance sizes it becomes the second ranked algorithm. However, for the largest computation time limit, the ranking of SAC worsens again. As observed in [12], these effects

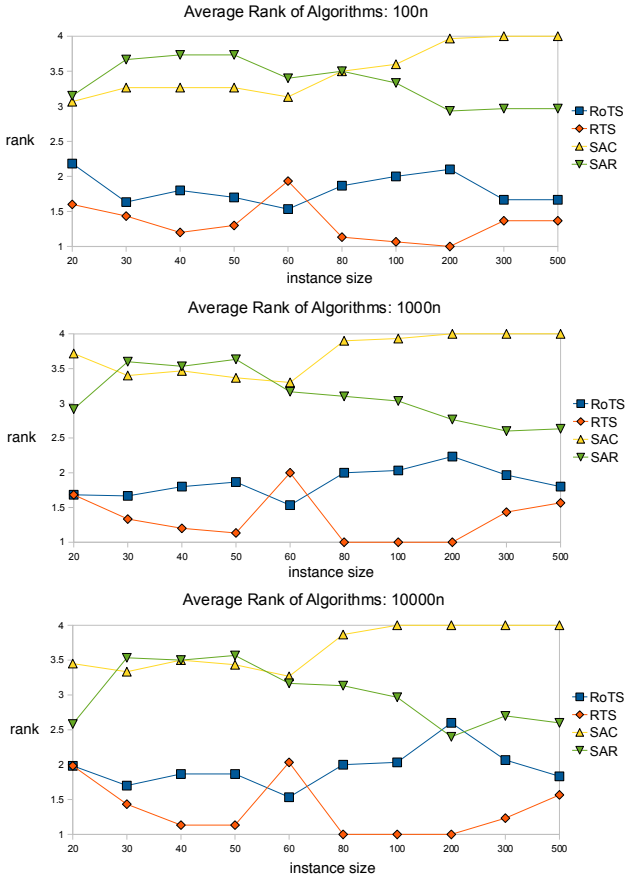

Figure 2: Line plots showing the average rank for RoTS, RTS, SAC, and SAR for RR instances at $100n(short)$, $1000n(medium)$, and $10000n(long)$.

may be due to the missing restart in the original version of SAC as proposed in [8]; this conjecture is also supported by the very good performance of SAR, which uses a temperature reheating mechanism that actually can be seen as a type of soft algorithm restart.

Given these in part surprising results, we were interested in considering strucutrally very different instance classes to examine whether analogous dependencies of the relative algorithm ranking on instance size occur. We therefore generated the additional sets of the RR and GR instances, which we already described in the previous section. Figure 2 shows the rank plots for RR instances, and Figure 3 for GR instances. On RR instances, the rank of both SA variants are clearly worse than those of the two TS variants across all instance sizes; only for the largest computation time limits, SAR appears to catch up with the TS algorithms. Between the two TS variants, RTS shows better results than RoTS across most instance sizes, which is consistent with the fact that RTS is known

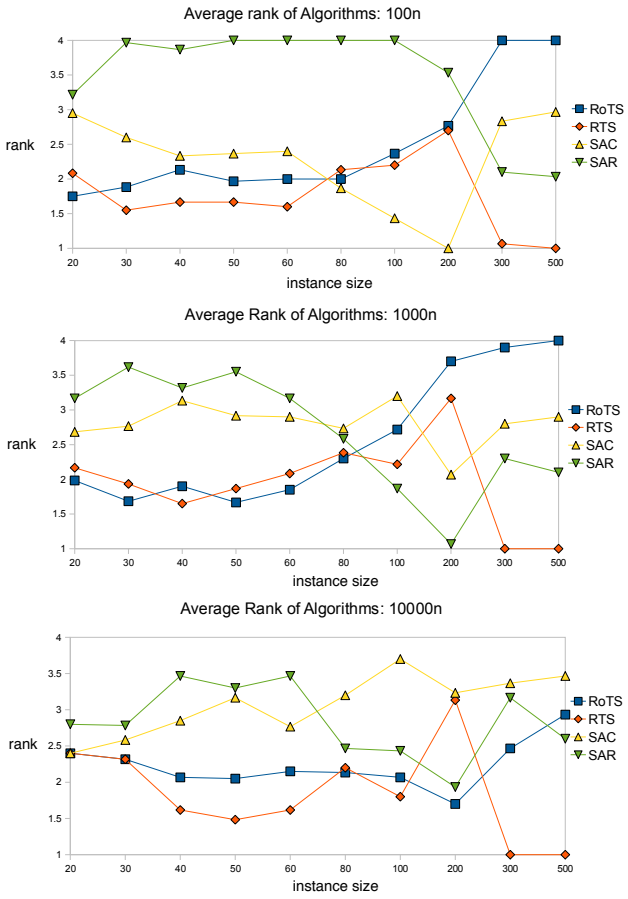

Figure 3: Line plots showing the average rank for RoTS, RTS, SAC, and SAR for GR instances at $100n(short)$, $1000n(medium)$, and $10000n(long)$.

to have excellent performance on instances such as those in the RR class. The results here confirm that the high performance of RTS also extraplotes to very large QAP instances of this instance class. The rank plots for GR instances show again some transitions in the relative performance of the algorithms for large instance sizes. RoTS and RTS show similar ranks up to instance size 100, while for larger instance sizes (in particular, 300 and 500) the rankings of the two tabu search algorithms diverge strongly. SAR becomes more competitive as instance size increases, while SAC is competitive with the other algorithms only for the short computation time limits.

5 Conclusions

In this article we have compared two versions of SA and two versions of TS algorithms. Our experimental design was targeted towards examining the relative performance of these algorithms in dependence of instance size. In fact, we found that the instance size may have a clear impact on which algorithm should be preferred. The strongest dependencies have been observed for instances that were generated to resemble the structure of real-life instances. On the contrary, for other instance classes, such as unstructured instances such as those of our RR class, TS algorithms were dominating the SA algorithms independent of instance size. Hence, whether the relative performance of algorithms changes with instance size also depends strongly on the particular class of instances.

There are a number of directions that may be interesting to investigate in future research efforts. A first one is to extend this experimental study to other classes of algorithms such as iterated local search, ant colony optimization, and memetic algorithms. Another direction is to examine the reasons for the changes in the relative performance of SA and TS algorithms in dependence of instance size. Finally, it may also interesting to examine more carefully the dependence of appropriate parameter settings in dependence of the instance size. In fact, obtaining an effective way of setting instance-szie dependent parameters is a widely open issue in metaheuristics.

Acknowledgements This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Mohamed Saifullah Hussin acknowledges support from the Universiti Malaysia Terengganu. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Research Associate.

References

[1] M. A. Arostegui, S. Kadipasaoglu, and B. M. Khumawala. An empirical comparison of tabu search, simulated annealing, and genetic algorithms for facilities location problems. *International Journal of Production Economics*, 103(2):742–754, 2006.

[2] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[3] R. Battiti and G. Tecchiolli. Simulated annealing and tabu search in the long run: A comparison on QAP tasks. *Computer and Mathematics with Applications*, 28(6):1–8, 1994.

[4] M. Birattari. On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs? Technical Report TR/IRIDIA/2004-001, IRIDIA, Université Libre de Bruxelles, Belgium, Brussels, Belgium, 2004.

[5] 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, Dordrecht, The Netherlands, 1998.

[6] E. Çela. *The Quadratic Assignment Problem: Theory and Algorithms*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[7] W.-C. Chiang and C. Chiang. Intelligent local search strategies for solving facility layout problems with the quadratic assignment problem formulation. *European Journal of Operational Research*, 106(2-3):457 – 488, 1998.

[8] D. T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46(1):93–100, 1990.

[9] 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.

[10] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657 – 690, 2007.

[11] C. Nugent, T. Vollman, and J. Ruml. An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research*, 16(1):150–173, 1968.

[12] G. Paul. Comparative performance of tabu search and simulated annealing heuristics for the quadratic assignment problem. *Operations Research Letters*, 38(6):577–581, 2010.

[13] J. Paulli. A computational comparison of simulated annealing and tabu search applied to the quadratic assignment problem. *Applied Simulated Annealing*, 46(1):85–102, 1993.

[14] M. Sinclair. Comparison of the performance of modern heuristics for combinatorial optimization on real data. *Computers & Operations Research*, 20(7):687–695, 1993.

[15] J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2(1):33–45, 1990.

[16] É. D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4–5):443–455, 1991.

[17] É. D. Taillard. Comparison of iterative searches for the quadratic assignment problem. *Location Science*, 3(2):87–105, 1995.

[18] J.-C. Wang. Solving quadratic assignment problems by a tabu based simulated annealing algorithm. In *Proceedings of 2007 International Conference on Intelligent and Advanced Systems, ICIAS 2007, Kuala Lumpur, Malaysia*, pages 75–80, 2007.