A Tabu Search Matheuristic for the Generalized Quadratic Assignment Problem

Published:
DOI: http://doi.org/10.1007/978-3-031-26504-4_46

A Tabu Search Matheuristic for the Generalized Quadratic Assignment Problem

Peter Greistorfer[1], Rostislav Staněk[2], and Vittorio Maniezzo[3]

[1] Operations and Information Systems, Karl-Franzens-Universität Graz, Graz, Austria
peter.greistorfer@uni-graz.at
[2] Applied Mathematics, Montanuniversität Leoben, Leoben, Austria
rostislav.stanek@unileoben.ac.at
[3] Computer Science, Università di Bologna, Bologna, Italy
vittorio.maniezzo@unibo.it

Abstract. This work treats the so-called Generalized Quadratic Assignment Problem. Solution methods are based on heuristic and partially LP-optimizing ideas. Base constructive results stem from a simple 1-pass heuristic and a tree-based branch-and-bound type approach. Then we use a combination of Tabu Search and Linear Programming for the improving phase. Hence, the overall approach constitutes a type of mat- and metaheuristic algorithm. We evaluate the different algorithmic designs and report computational results for a number of data sets, instances from literature as well as own ones. The overall algorithmic performance gives rise to the assumption that the existing framework is promising and worth to be examined in greater detail.

Keywords: Generalized Quadratic Assignment · Matheuristic · Metaheuristic · Linear Programming · Tabu Search.

1 Introduction

The problem of interest is the so-called *Generalized Quadratic Assignment Problem* (GQAP). The GQAP has been used as a model for several relevant actual applications, including order picking and storage layout in warehouse management, relational database design or scheduling activities in semiconductor wafer processing. Technically, it originates from the *Linear Assignment Problem* (LAP), where a number of agents (equivalently, machines or supplies) have to be assigned to a number of jobs (tasks or demands), while minimizing the total cost of service and obeying assignment constraints, which secure that each job has to be serviced by exactly one agent and vice versa. The LAP in turn is a special case of the *Generalized Assignment Problem* (GAP), in which assignment constraints on the supply-side are replaced with upper bound constraints, which model an agent's capacity consumed by the assigned task weights. Another generalization of the LAP is the *Quadratic Assignment Problem* (QAP), which models multiplicative cost factors between agents and jobs, e.g. in locational analyzes distances times interaction frequencies. Finally, the GQAP can be thought of as a combination of the GAP and the QAP, where the objective function receives a quadratic component in addition to the linear one, the same way as it is done in the QAP and suppliers have an upper bound constraint as in the GAP.

It is well-known that there exist efficient polynomial algorithms for the LAP, e.g. the Hungarian method, but the GAP, the QAP, and the GQAP are $\mathcal{NP}$-hard (e.g. see [1]). Matheuristics (see [6, 8]), a synthesis of classical, as a rule, (real-valued) linear and integer linear programming methods (LP, ILP) with conventional heuristic methods (e.g. local search) and/or modern metaheuristic methods (Tabu Search, GRASP, Scatter Search etc.) have become popular with the rise of recent powerful hardware and even more because of the success of solvers like CPLEX or Gurobi.

Our research focuses on both mathematical formulations of the GQAP as well as on the development of LP- and ILP- based matheuristics. In this paper we review a branch-and-bound type heuristic tree search and elaborate on the basic decomposition idea of the LP-component in the improvement algorithm. Afterwards, the focus lies on the presentation of a new Tabu Search (TS), the *TS-matheuristic GQAP* approach (`TS-GQAP`) is presented. We conclude with current computational results and a short outlook.

2 Modelling the GQAP

The GQAP can be described by means of the following quadratic integer program. We are given $m \in \mathbb{N}$, the number of agents, $n \in \mathbb{N}$, the number of jobs, with linear assignment costs and weights $p_{ij} \in \mathbb{R}$ and $w_{ij} \in \mathbb{R}_0^+$, where $1 \leq i \leq m$ and $1 \leq j \leq n$, respectively. The weights present resource amounts to be spent by agent i for processing job j, without exceeding an available capacity $a_i \in \mathbb{R}_0^+$. Quadratic costs are defined by the product of $d_{ir} \in \mathbb{R}$, the cost factor between the agents i and r, and $f_{js} \in \mathbb{R}$, the cost factor between the resources j and s. Binary decision variables $x_{ij} \in \{0,1\}$ determine whether a job j is served by agent i, or not. Then the GQAP is defined by the following binary quadratic program (BQP):

$$\min \sum_{i=1}^{m} \sum_{j=j}^{n} p_{ij} x_{ij} + \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{r=1}^{m} \sum_{s=1}^{n} d_{ir} f_{js} x_{ij} x_{rs} \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_{ij} x_{ij} \leq a_i \qquad \forall\ 1 \leq i \leq m, \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \forall\ 1 \leq j \leq n, \tag{3}$$

$$x_{ij} \in \{0,1\} \quad \forall\ 1 \leq i \leq m,\ \forall\ 1 \leq j \leq n. \tag{4}$$

3 Solving the GQAP with TS-GQAP

3.1 Basic concepts

We start with a short summary of the constructive solution procedure, which was in its original form presented as *Guided Adaptive Relaxation Rounding Procedure* (`GARRP`) in [7]. `GARRP` starts with an empty (infeasible) solution and iteratively creates partial solutions by fixing exactly one agent-task-assignment $x'_{ij} := 1$ at each iteration, based upon the optimum solution of a relaxation of BQP, leaving a subset of remaining decision variables as free. These partial solutions, X', i.e. incomplete or partial assignments of jobs j to single specific agents i, are stored within a tree, in which each tree-node represents one assignment made. So for any (partial) solution all fixed assignments left can be found by pegging links back to the root, which stores the result of the first relaxation solved (compare [4]). At any node, relaxed, i.e. continuous values x^*_{ij} received from a GQAP-LP-solution, define choice-probabilities among the k-best possible succeeding fixations $x^*_{ij} \geq x_{LB}$ (k and x_{LB} being parameters), from which the most likely option is chosen. Parameters k and x_{LB} are user inputs. The whole tree is explored in a depth-first manner, if needed investigating all potential successor assignments according to a width-second sequence. If at some node all successor options are unsuccessfully investigated (in terms of feasibility defined by capacity constraints), a backtracing process starts until a node with a free successor, becoming investigated, is found or until all nodes failed because of overall infeasibility. Normally, this iterative process stops as soon as n assignments have been made with a complete and feasible solution, otherwise it fails.

The present work, focusing on improvement methods for the GQAP, has its roots in the *Magnifying Glass Heuristic* approach, which was already successfully applied to the *Quadratic Travelling Salesman Problem* in [10]. Then, for the first time, this approach was adopted as `MG-GQAP` [2] to improve (heuristic) GQAP solutions. As might be expected, while the results were quite satisfying, it turned out that there was still room for improvement. First, we considered the possibility to include the quadratic components into the linear part by compressing the quadratic coefficient matrix into a one-dimensional vector, by means of techniques borrowed from data dimensionality reduction approaches, like principal component analysis. Specifically, we computed when possible (it has always been) the eigenvalues of the coefficient matrix and used them in our heuristic. The resulting code was efficient in the GQAP part, but the computation of the eigenvalues was very demanding in case of big instances, moreover local search was used in the method and it appeared as if the method was apparently not sufficiently capable to escape from local optima. So, the idea was to dismiss linearizations and, sort of the other way round, to use linear programming as an improvement to a TS. This led to the design of the `TS-GQAP`, which combines two distinct parts, the TS-neighbourhood and the `MG-GQAP`.

For the TS and its neighbourhood the performance of two prominent standard operators was analyzed: either two jobs swap machines (*exchange*) or a job changes to another agent (*insert*). We contrasted using these two operators together versus using them specifically. And since exclusive use of insert-moves turned out to work best, this setting was chosen. In the course of the exploration the TS sequentially checks all potential re-assignments of tasks. In doing so, a neighbourhood of admissible TS_nh_size trial moves is build by using a short-term recency tabu-memory. This memory is defined on the time, i.e. iteration, when a job is assigned to a new agent. Using this information, a newly positioned task is not allowed to be removed from its agent for a parametric number of *tenure* iterations. As aspiration criterion, the best-improvement-rule is used, in which case the tabu status is released from a given trial move if it improves the incumbent best solution.

`MG-GQAP` can be seen as a variable fixing heuristic, which decomposes the overall problem into a fixed and a free component. Thus, interpreting release as destruction, the method shares similarities with a *Large Neighbourhood Search*, where solutions are partially destroyed and repaired by corresponding operators; a process, which is iteratively repeated until a local optimum is reached (see [9]). For the GQAP the corresponding idea works like this: according to pre-defined selection patterns, we consider K chosen columns (jobs, in constraints (3) of BQP) and create a new auxiliary instance containing only these K columns (and all rows). Firstly, linear costs p_{ij} are modified including the quadratic costs caused by relations between the assignment of j to i and the already fixed assignments outside of the chosen columns. Then, after re-adjusting the total amounts of resources a_i in order to reflect the remaining capacities, we solve the auxiliary problem optimally and get a new, possibly improved solution with changes restricted to the K chosen columns. Note that alternatively also subsets of *rows* (agents, constraints (2)) may be selected to define new auxiliary subproblems, which establishes the differentiation between a column- and a row-oriented variant of `MG-GQAP`. The overall process ends after a total number of given iterations. It should be noted that the idea behind `MG-GQAP` is even capable of serving as a construction procedure. In that case it starts from an empty unfeasible solution, iteratively increasing the size of a partial solution as long as assignments are found, which were not prevented by binding capacity restrictions, while ultimately and ideally constructing a full feasible solution with n assignments. We give this simple 1-pass start heuristic the name `MG-GQAP-C` and report results in the computational section.

In summary, within `TS-GQAP`, the TS utilizes the LP-part in order to optimize a good local solution in terms of intensification. The role of diversification is taken over by the occasional use of elite solutions collected in a *pool* by the TS. Section 3.2 explains this component and covers the algorithmic details.

3.2 Implementation details

It is well-known that metaheuristic developments have proved to be successful especially in cases, where their fundamental concepts are complemented with pool-oriented approaches, originally rooted in Genetic Algorithms or Scatter Search and Path Relinking. These algorithms maintain a reference set of high quality solutions, which are repeatedly used during the search in order to guarantee a fruitful balance between diversification and intensification (see, e.g. [3]). Therefore, Algorithm 1: TS-GQAP uses a pool structure as follows.

Require: a GQAP instance with a solution $X := [x_{ij}]^{m \times n} \in \{0,1\}^{m \times n}$
Ensure: new solution X_{new} with $c(X_{new}) \leq c(X)$
1: $X^* := X$, $c_{best} := c(X^*)$
2: **repeat**
3:     **repeat** // TS-phase:
4:         $X' := best\_admissible\big(TS\_neighbourhood(X)\big)$
5:         **if** $c(X') < c_{best}$ **then**
6:             update $X^* := X'$
7:             $c_{best} := c(X')$
8:         **end if**
9:         maintain a set *pool* of good and diverse solutions
10:     **until** $TS\_termination$
11:     **if** TS_failed **then**
12:         $X' := unchecked\_solution(pool)$
13:     **end if**
14:     **repeat** // MG-GQAP-phase:
15:         $Col := select\_variable\_columns(X')$
16:         $X_{LP} := LP\_from(Col)$
17:         $X^*_{LP} := solve(X_{LP})$ // CPLEX
18:         $X'' := combine\_solutions(X', X^*_{LP})$
19:         **if** $c(X'') < c_{best}$ **then**
20:             update $X^* := X''$
21:             $c_{best} := c(X'')$
22:         **end if**
23:     **until** $MG\_QAP\_termination$
24:     $X := X''$
25: **until** $TS\_GQAP\_termination$
26: **return** $x_{new} := X^*$

Algorithm 1: TS-GQAP

As an improvement procedure, the `TS-GQAP` builds on a feasible solution X with objective function value $c(X)$. It consists of alternating `TS-` and `MG-GQAP-` phases (lines 3 – 10 and 14 – 23), i.e. it is a series of `TS1, MG-GQAP1, TS2, MG-QAP2, ...` until an overall termination criterion, $TS\_GQAP\_termination$, gets true. Each `TS` is capable of maintaining the pool of elite solutions. Such solutions are collected to be used in future iterations of the search process. Solution X' in line 4 is the actual and best admissible solution iteratively drawn from consecutive TS-neighbourhoods, which involves the maintenance and usage of the TS-memory as described above.

The pool maintained by the TS, in line 9, collects a maximum of $pool\_size$ good and diverse solutions. In doing so, a solution is deemed good at the moment, when it has just been improved by the TS, i.e. goodness means objective function value and diversity targets the structural difference mapped in the values of the decision variables, i.e. assignments. In the current version such diversity is – at least with higher probability – ensured by allowing into the pool only members with different objective function values. The diversification step is done in lines 11 – 13 as soon as the TS was not able to improve the current solution, expressed by a value *true*, returned from function TS_failed. To diversify the search, function $unchecked\_solution(pool)$ extracts the cost-minimum solution from the pool, which has not yet been output from the pool to be processed by the `MG-GQAP`. In the case that all pool solutions are already processed, this procedure forwards the running, i.e. the best solution from the last `TS`-phase to the `MG-GQAP`. For function $select\_variable\_columns(X')$ in line 15, which stores free assignments to be optimized by an LP in set Col, we considered a number of possibilities. For the determination of these $K := |Col|$ columns, several selection mechanisms were tested. These include *random* selection, so-called *plane* selection of columns ((1,2,3), (4,5,6), ..., (n - 2, n - 1, n); (2,3,4), (5,6,7),...) or *binomial* selection with columns ((1), (2), (3), (1,2), (1,3), (2,3), (1,2,3,); (2), (3), (4),...), all examples underlying $K = 3$. Trends in efficiency are non-random strategies $\prec$ purely random $\prec$ mixed-random options. The chosen strategy is a half and a half mixture of *random* and *plane*.

After building the LP-subproblem by $LP\_from(Col)$ and solving it, in lines 16 and 17, respectively, procedure $combine\_solutions(X', X^*_{LP})$ adds the optimized values $x^*_{ij_{LP}}$ to the starting solution X' and thus includes the LP-optimized assignments of X^*_{LP}. If necessary, this is followed by an update of the best solution, the same way as it is done fo the TS in lines 5 – 8. At the very end, with $TS\_GQAP\_termination$ getting true, the overall process stops and the algorithm returns its best solution found as X_{new}.

4 Computational results

For the evaluation of the computational results we used three test beds with a total of 64 instances. Abbreviated with LAM, CEAL and OWN, there are 27 instances with $m = 6 - 30$ and $n = 10 - 16$ from Lee and Ma [5], 21 instances with $m = 6 - 20$ and $n = 20 - 50$ from Cordeau et al. [1] and 16 own randomly generated instances with $m, n = 10 - 200$. All algorithms were implemented in AMPL script V.20220310, using the solver CPLEX V.20.1.0.0 (MS VC++ 10.0, 64-bit). All runs were performed on a ThinkPad X1 notebook with an Intel(R) Core(TM) i7-8550U CPU @1.80GHz (Aug. 2017) under Windows 10 Pro with 8 GB RAM.

We aimed for a unified parameter setting, but eventually this was only possible to a certain extent due to the large structural differences in the data sets: symmetry of the data matrices D and F, constancy of weights in W over the agent set and, very basic, instance sizes, where the latter directly affect and determine calculable LP subproblems, regarding the value of parameter K. To profit from faster calculations, a relatively low value of $TS\_nh\_size = 3$ is chosen. Tabu time becomes $tenure \approx 0.3n$ and $pool\_size = 20$. More numeric parametric details are reported below. Evaluation is split into two parts, judgement of the general design of the construction and improvement procedures, thereafter in part two supplemented by the discussion of the specific results obtained for the three data sets.

In order to test and evaluate the general design of the construction procedures, we used all 3 tests sets. Starting with the tree heuristic `GARRP` and LAM, parameter settings were $x_{LB} = 0.5$ and $k = 50$. The procedure needs $min|avg|max = 6|2375|18364$ tree nodes and $min|avg|max = 0.3|168.5|1598.4$ seconds (sec.). The second test set, CEAL, is solved with $x_{LB} = 0.85, k = 25$ and $min|avg|max = 21|1842|16103$ nodes, $min|avg|max = 1.1|391.6|\underline{3600}$ sec. Note that while feasible solutions for all instances of LAM can be built, the method fails in two cases (#11 and #15) for CEAL, i.e. those ones for which one hour of computing time was not enough. Here starting solutions can be provided by solving the standard (linear) GAP. Results for set OWN, solved with $x_{LB} = 0.9, k = 30$, are $min|avg|max = 11|30.6|201$ nodes and $min|avg|max = 0.3|65.3|707.1$ sec. Again, as in LAM, no infeasible solution had to be accepted. Not much can be derived from these numbers, but two statements appear to be meaningful: the number of nodes, equivalent to the number of solved LPs, is relatively big for specific instances, which indicates a large CPU load. Secondly, it is a pleasing fact that the success rate of the procedure is quite high (62 of 64 cases). An objective-oriented reasoning of `GARRP` goes along with the judgement of the magnifying glass approach used as a construction procedure, i.e. algorithm variant `MG-GQAP-C`. This time, exclusively based on CEAL, setting $K = 4$, three `MG-GQAP-C`-runs with $MG\_QAP\_term = 500|1000|10000$ iterations are performed. The hypothesis is that with an increasing number of iterations, i.e. with higher computing time, the numerical quality of the overall best solution will increase too, though by an unknown and decreasing amount. These expectations are met as follows. Designating the result of the 500-run as a basis, the relative stepwise improvements obtained between the 500- and the 1000-result and between the 1000- and the 10000-result, averaged over the objective values of all 21 instances, are 2.7% and 0.8%. Moreover, contrasting the best `MG-GQAP-C`-result, the 10000-result with `GARRP`, it becomes obvious that the tree-procedure works even better with an averaged additional 3.9% performance gain. It is interesting to note that these numerical ratios apply not only for the construction process but also for an improvement process that builds exactly on the former starting solutions. Total running time (min.) naturally gradually increases: 14.2|33.0|80.6 (`MG-GQAP-C`) vs. 137.1 (`GARRP`).

Next, we appraise the design of the proposed TS improvement procedure `TS-GQAP`. Essentially, it consists of a TS component (TS-phase) and an LP component (MG-GQAP-phase). So it is obvious to isolate the individual components and to compare three test runs: (1) only MG-GQAP-phase, (2) only TS-phase and (3) both phases combined (=`TS-GQAP`). Like above, test set CEAL is used, input comes in all cases from the tree start heuristic `GARRP` and parametrization from a unified, single parameter set. The outcome for (1) is an average improvement of $\delta_{tree}^{avg} = 5.75\%$ (in 35.1 min., whole set), for (2) it is 2.5% (38.0 min.)

and finally for (3), the `TS-GQAP`, it is 6.39% (44.1 min.). Thus, even if the overall concept gets sufficiently motivated, some captious comments seem appropriate. Again, as an logical implication, objective function improvement comes at the expense of CPU time. In the present context, however, the contribution of the TS-phase is smaller than that of the MG-GQAP-phase. However, this is not surprising since the optimizing component based on an optimum solution algorithm will generally generate more visible improvements than a heuristic one. One can also see that the neighbourhood structure of the TS, which is more complex in terms of implementation, has a significant impact. Nonetheless, the contribution of the TS is also a significant one, which is only underscored by the overall effectiveness of the method.

The second part of the computational results covers the evaluation of the specific results calculated for the three test sets. Moderately sized instances of LAM allow the use of a mixed strategy. The number of LP-columns as well as the number of LP-rows is set with $K = 4$ and the LP-build strategy, column or row-oriented, is changed every 25 iterations. Termination parameters are set as $TS\_term = 150, MG\_GQAP\_term = 50$ and $TS\_GQAP\_term = 20$. The result for LAM is $\delta^{avg}_{tree} = 11.66\%$, which takes 2.4 min. averaged over all instances. It can be observed that the algorithm finds the best solutions quite early. With respect to quality it can be stated that the results appear to be good, but no optimality gaps were calculated since optimal solutions are not published.

The next object of observation is the test set CEAL. Again, $K = 4$ and the LP-component follows a column approach, exactly as it is described in Algorithm 1, while completion criteria are given by $TS\_term = 500, MG\_GQAP\_term = 100$ and $TS\_GQAP\_term = 10$. This leads to an improvement of $\delta^{avg}_{tree} = 6.4\%$ in an average of 2.2 min. Because competing objective function values are available, deviations from best-known objectives can be determined. They are given with δ_{CEAL} and amount $min|avg|max = 0.0\%|1.4\%|8.29.\%$. Results' quality clearly correlates with instance-density ρ, the ratio of total capacity demanded over total capacity available (reasonably only calculable for server independent constant demands, as it is the case with CEAL and LAM). As already indicated the two problem instances, namely #11 and #15, cannot be improved. It should also be stated in an exculpatory manner that average quality gets destroyed by only a few outliers. Compensating these, an acceptable $\delta^{avg}_{CEAL} = 0.9\%$ can be achieved.

The third and last set, OWN, contains the hardest instances: asymmetric, non-constant weights and sizes up to $m, n = 200$. It is solved with the same column-oriented LP-strategy as used for CEAL. We set $K = 3$, however, due to advanced dimensions, it was necessary to reduce the value of K to 2 for instances with a high $m = 200$. Termination variables are given by $TS\_term = 150, MG\_GQAP\_term = 25$ and $TS\_GQAP\_term = 40$. With these parameters we can achieve a $\delta^{avg}_{tree} = 34.46\%$ in average 11.0 min. In the OWN case running time utilization is more efficient since for some instances the best result is only achieved after 90% of the total running time. The majority of instances ($\approx 11, 12$ out of 16) cannot be solved optimally with the means at our disposal, hence no reasonable deviations can be calculated.

As a final remark it should be noted that for all test sets, LAM, CEAL and OWN, algorithm `TS-GQAP` constitutes an improvement over the old `MG-GQAP` as described in [2]. In terms of objective value amount this progress is not outstanding, but it is clearly visible. It comes either as an increase in CPU productivity, i.e. percentage improvement divided by CPU time used, or actually as an improvement of the (average) objective.

5 Summary, criticism and outlook

This work is a further step in an ongoing research project looking for heuristic solution procedures for the GQAP. We combine the well-known metaheuristic TS with the strengths of mathematical programming and introduce a new matheuristic referred to as `TS-GQAP`. The basic idea behind it was originally coined as Magnifying Glass Heuristic, itself a matheuristic and successfully used for a quadratic *Travelling Salesman Problem*. The design of the new method turned out to be successful in terms of CPU usage and the ability to deal with larger problem sizes, while also specific results could be improved. Moreover, in terms of competition, the method proved to be able to keep up with algorithms from literature and the best-known-gaps could be reduced by another level.

Even if the new algorithmic design is promising, there exist clearly visible improvement opportunities. New challenges raised are those about the course of the interaction between the TS neighbourhood and the LP decomposition or about the implementation of an overarching memory structure. Very basically, a stumbling block on the way to outstanding performance is founded in the capacity restricted nature of the underlying problem, which logically is intricate to navigate. Fast metaheuristics are able to play off their superiority for problem classes, which are unrestricted or endowed with a large number of feasible solutions (dense solution space) and often benefit from easier objective function calculations. As observed, the explorable solution space, more accurately, the neighbourhood induced solution landscape for some GQAP instances investigated is quite sparse. It has already been put forward [6] that sparse solution spaces are indicative of cases where matheuristics are probably more effective than plain metaheuristics, usually relying on local search or simple constructive procedures at their core. Moreover, GQAP is a representative problem of nonlinear combinatorial optimization, an area that so far received much less attention from research than its linear counterpart, despite its obvious relevance for modelling and solving compelling real-world problems.

This opens sufficient room to set up and tune new neighbourhood mechanisms to be developed, an endeavor, which of course cannot be done apart from designing more efficient memory structures. It is precisely this problem area that must be examined and analyzed more closely in the future.

References

1. Cordeau, J.F., Gaudioso, M., Laporte, G., Moccia, L.: A memetic heuristic for the generalized quadratic assignment problem. INFORMS Journal on Computing **18**, 433–443 (11 2006). https://doi.org/10.1287/ijoc.1040.0128
2. Greistorfer, P., Staněk, R., Maniezzo, V.: The magnifying glass heuristic for the generalized quadratic assignment problem. In: Proceeding of the XIII Metaheuristics International Conference MIC 2019. pp. 22–24. Universidad de los Andes Sede Caribe (2019)
3. Greistorfer, P., Voß, S.: Controlled Pool Maintenance for Metaheuristics, pp. 387–424. Springer US, Boston, MA (2005). https://doi.org/10.1007/0-387-23667-8_18
4. Kaufman, L., Broeckx, F.C.: An algorithm for the quadratic assignment problem using Bender's decomposition. European Journal of Operational Research **2**(3), 207–211 (1978)
5. Lee, C., Ma, Z.: The generalized quadratic assignment problem. Research Report, Department of Mechanical and Industrial Engineering, University of Toronto, (2003)
6. Maniezzo, V., Boschetti, M., Stützle, T.: Matheuristics: Algorithms and Implementations. EURO Advanced Tutorials on Operational Research, Springer International Publishing (2021)
7. Maniezzo, V., Greistorfer, P., Staněk, R.: Exponential neighborhood search for the generalized quadratic assignment problem (2018), eURO/ALIO International Conference on Applied Combinatorial Optimization, 25.–27.6.2018, Bologna, Italy
8. Maniezzo, V., Stützle, T., Voß, S.: Matheuristics: Hybridizing Metaheuristics and Mathematical Programming. Springer Publishing Company, Incorporated, 1st edn. (2009)
9. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science **40**(4), 433–443 (2006)
10. Staněk, R., Greistorfer, P., Ladner, K., Pferschy, U.: Geometric and LP-based heuristics for the quadratic travelling salesman problem. Computers & Operations Research **108**, 97–111 (2019)