Consultant-Guided Search Algorithms for the Quadratic Assignment Problem

Serban Iordache

SCOOP Software GmbH, Cologne, Germany
siordache@acm.org

Abstract. Consultant-Guided Search (CGS) is a recent swarm intelligence metaheuristic for combinatorial optimization problems, inspired by the way real people make decisions based on advice received from consultants. Until now, CGS has been successfully applied to the Traveling Salesman Problem. Because a good metaheuristic should be able to tackle efficiently a large variety of problems, it is important to see how CGS behaves when applied to other classes of problems. In this paper, we propose an algorithm for the Quadratic Assignment Problem (QAP), which hybridizes CGS with a local search procedure. Our experimental results show that CGS is able to compete in terms of solution quality with one of the best Ant Colony Optimization algorithms, the MAX-MIN Ant System.

Keywords: Metaheuristics, combinatorial optimization, swarm intelligence, quadratic assignment problem.

1 Introduction

Consultant-Guided Search (CGS) [10] is a swarm intelligence [3] technique based on the direct exchange of information between individuals in a population. It takes inspiration from the way real people make decisions based on advice received from consultants. CGS is a metaheuristic that can be applied to virtually any hard combinatorial optimization problem, but until now it has been evaluated only on the Traveling Salesman Problem (TSP). For this class of problems, experimental results have shown that CGS is able to outperform some of the best Ant Colony Optimization (ACO) [7] algorithms. Nevertheless, the question remains if CGS is still able to achieve high performance when applied to other combinatorial optimization problems.

In this paper, we introduce CGS-QAP, an algorithm for the Quadratic Assignment Problem (QAP), which hybridizes CGS with a local search procedure. In our experiments we use MAX-MIN Ant System (MMAS) [15] as a yardstick to compare the performance of the proposed algorithm. The experimental results show that the solution quality obtained by CGS-QAP is comparable with or better than that obtained by MMAS.

In the present work, we also discuss how CGS relates to other metaheuristics for combinatorial optimization. We argue that CGS is a hybrid metaheuristic, by identifying a series of concepts borrowed from other optimization techniques. This is an important aspect that has not been addressed in the original CGS paper [10].

The rest of this paper is organized as follows: in order to make the paper self-contained, we describe in Section 2 the CGS metaheuristic; in Section 3 we place CGS in the context of heuristic optimization methods; in Section 4 we introduce the CGS-QAP algorithm and we report our experimental results; in Section 5 we conclude the paper and present future research directions.

2 The CGS Metaheuristic

CGS is a population-based method. An individual of the CGS population is a virtual person, which can simultaneously act both as a client and as a consultant. As a client, a virtual person constructs at each iteration a solution to the problem. As a consultant, a virtual person provides advice to clients, in accordance with its ***strategy***. Usually, at each step of the solution construction, there are several variants a client can choose from. The variant recommended by the consultant has a higher probability to be chosen, but the client may opt for one of the other variants, which will be selected based on some heuristic.

At the beginning of each iteration, a client chooses a consultant based on its ***personal preference*** and on the consultant's ***reputation***. The reputation of a consultant increases with the number of successes achieved by its clients. A client achieves a ***success***, if it constructs a solution better than all solutions found until that point by any client guided by the same consultant. Each time a client achieves a success, the consultant adjusts its strategy in order to reflect the sequence of decisions taken by the client. The exact details of how reputation and personal preference are used in order to select a consultant are specific to each application of CGS to a particular class of problems.

Because the reputation fades over time, a consultant needs that its clients constantly achieve successes, in order to keep its reputation. If the consultant's reputation sinks below a minimum value, it will take a ***sabbatical leave***, during which it will stop offering advice to clients and it will instead start searching for a new strategy to use in the future.

The pseudocode that formalizes the CGS metaheuristic is shown in Fig. 1. During the initialization phase (lines 2-5), virtual people are created and placed in sabbatical mode. Based on its mode, a virtual person constructs at each iteration (lines 7-33) either a solution to the problem (line 13) or a consultant strategy (line 9). Optionally, a local optimization procedure (line 17) may be applied to improve this solution or consultant strategy.

After the construction phase, a virtual person in sabbatical mode checks if it has found a new best-so-far strategy (lines 20-22), while a virtual person in normal mode checks if it has achieved a success and, if this is the case, it adjusts its strategy accordingly (lines 24-29). At the end of each iteration, the reputation and action mode of each virtual person are updated (lines 32-33).

```
 1. procedure CGSMetaheuristic()
 2.    create the set $\mathcal{P}$ of virtual persons
 3.    foreach $p \in \mathcal{P}$ do
 4.       setSabbaticalMode(p)
 5.    end foreach
 6.    while(termination condition not met) do
 7.       foreach $p \in \mathcal{P}$ do
 8.          if actionMode[p]=sabbatical then
 9.             currStrategy[p] ← constructStrategy(p)
10.          else
11.             currCons[p] ← chooseConsultant(p)
12.             if currCons[p] ≠ null then
13.                currSol[p] ← constructSolution(p, currCons[p])
14.             end if
15.          end if
16.       end foreach
17.       applyLocalOptimization()                    // optional
18.       foreach $p \in \mathcal{P}$ do
19.          if actionMode[p]=sabbatical then
20.             if currStrategy[p] better than bestStrategy[p] then
21.                bestStrategy[p] ← currStrategy[p]
22.             end if
23.          else
24.             c ← currCons[p]
25.             if c ≠ null and currSol[p] is better than all solutions
26.                         found by a client of c since last sabbatical then
27.                successCount[c] ← successCount[c]+1
28.                strategy[c] ← adjustStrategy(c, currSol[p])
29.             end if
30.          end if
31.       end foreach
32.       updateReputations()
33.       updateActionModes()
34.    end while
35. end procedure
```

Fig. 1. The CGS Metaheuristic

Fig. 2 details how consultants' reputations are updated based on the successes achieved by their clients. Each consultant is ranked based on the best result obtained by any client working under its guidance. For a number of top-ranked consultants, CGS prevents their reputations from sinking below a predefined level (lines 12-14).

Fig. 3 details how the action mode of each virtual person is updated: consultants whose reputations have sunk below the minimum level are placed in sabbatical mode, while consultants whose sabbatical leave has finished are placed in normal mode.

```
procedure updateReputations()
    foreach $p \in \mathcal{P}$ do
        if actionMode[p]=normal then
            rep[p] ← applyReputationFading(rep[p])
            rep[p] ← rep[p] + successCount[p]
            if currSol[p] is better than best-so-far solution then
                rep[p] ← rep[p] + bonus
            end if
            if rep[p] >maxReputation then
                rep[p] ← maxReputation
            end if
            if isTopRanked(p) then
                rep[p] ← enforceMinimumReputation(rep[p])
            end if
        end if
    end foreach
end procedure
```

Fig. 2. Procedure to update reputations

```
procedure updateActionModes()
    foreach $p \in \mathcal{P}$ do
        if actionMode[p]=normal then
            if rep[p] <minReputation then
                setSabbaticalMode(p)
            end if
        else
            sabbaticalCountdown ← sabbaticalCountdown - 1
            if sabbaticalCountdown = 0 then
                setNormalMode(p)
            end if
        end if
    end foreach
end procedure
```

Fig. 3. Procedure to update action modes

Fig. 4 shows the actions taken to place a virtual person in sabbatical or in normal action mode.

3 Positioning of CGS

CGS introduces a new metaphor, but it is not obvious whether it represents a novel metaheuristic or rather a reformulation of a known method, using new names for existing concepts. In this section, we try to place CGS in the context of heuristic optimization methods and we argue that it represents a hybrid metaheuristic, which combines new ideas with concepts found in other optimization techniques.

```
1. **procedure** setSabbaticalMode(p)
2.     actionMode[p] ← sabbatical
3.     bestStrategy[p] ← null
4.     sabbaticalCountdown ← sabbaticalDuration
5. **end procedure**

6. **procedure** setNormalMode(p)
7.     actionMode[p] ← normal
8.     rep[p] ← initialReputation
9.     strategy[p] ← bestStrategy[p]
10. **end procedure**
```

Fig. 4. Procedures to set the sabbatical and normal mode

CGS can be classified as a model-based search (MBS) algorithm [19]. In MBS, candidate solutions are constructed using some parameterized probabilistic model. These solutions are then used to modify the probabilistic model in order to bias future sampling toward high quality solutions. CGS mainly differs from other MBS algorithms like ACO and estimation of distribution algorithms (EDA) [12] in the way it bias probabilities with respect to past experience in order to intensify the search around the best combinations. Instead of using an indirect pheromone-based learning mechanism (like in ACO), or estimation of distributions (like in EDA), a set of combinations is used in CGS to directly bias probabilities.

Somewhat surprisingly, CGS can even be cast into the formal framework of ant programming (AP) [2]. AP is based on the use of an iterated Monte Carlo approach for the multi-stage solution of combinatorial optimization problems. A population of agents, called ants, is used in order to construct solutions. Each agent perceives the state of the system through a representation, which can be seen as a mental image and, in general, gives less information than the state description. During the solution construction, each ant moves on the state graph, but it represents its movement on the representation graph. At each step, a set of feasible candidate actions is determined based on information pertaining to the system state. One of these actions is then selected based on a probabilistic policy parameterized in terms of a desirability function. The desirability function associates a real value to each edge of the representation graph. For both ACO and CGS, a partial solution is expressed as a sequence of components, while the AP representation of a state can be expressed as the last component added to the partial solution. In the case of ACO, the desirability is given by the amount of pheromone on the edge connecting the last component included and the candidate component. In the case of CGS, the desirability function can be expressed in terms of two elements: the reputations and the strategies of the consultants. The reputations bias the probability to use a given strategy, which in turn biases the probability to select a given component. In AP, the desirability information is updated on the basis of the cost of the generated solutions. CGS fits into the AP framework, because both reputations and strategies are updated in accordance with the successes achieved by clients. This shows that the AP framework is not restricted to pheromone-based algorithms, but it can also accommodate algorithms that use non stigmergic communication.

In CGS, the information exchange is based on direct communication, thus bearing some resemblance to bee inspired algorithms [11]. These algorithms mimic the behavior of real bees, which perform a so-called waggle dance in order to transmit information about the direction and distance to a food source. In this way, a bee is able to recruit other nest mates to the discovered food source. In the CGS metaphor the consultants wait passively to be selected by clients, but we can consider that they actually try to recruit clients. The recruitment procedure in CGS differs though from the recruitment procedure in bee inspired algorithms in the way the probability to recruit an agent is biased: CGS uses the consultant reputation, while most bee inspired algorithms use the solution quality.

CGS combines several concepts found in other optimization techniques. For example, the reputation fading is similar to the pheromone evaporation in ACO. Another example is the construction of a new strategy during the sabbatical leave. This process resembles the escape mechanism used in Reactive Tabu Search [1] when the system is trapped in a *complex attractor*, or the pheromone trail reinitialization performed in MAX-MIN Ant System [15] when the algorithm approaches the stagnation behavior. As a final example, we consider how CGS keeps information about promising solutions, by means of consultant strategies. This approach of maintaining a list of high quality solutions can also be found in some ACO variants like Population-Based ACO [8] or $ACO_{\mathbb{R}}$ [13].

4 Applying CGS to the QAP

In this section, we discuss how CGS can be applied to the QAP and we propose the CGS-QAP algorithm, which hybridizes CGS with a local search procedure. Then, we describe the experimental setting and we compare the performance of CGS-QAP with that of MAX-MIN Ant System.

4.1 The CGS-QAP Algorithm

Given n facilities and n locations, a flow matrix $F = \{f_{ij}\}$ and a distance matrix $D = \{d_{ij}\}$, the QAP consists in finding an assignment ϕ of facilities to locations, which minimizes the cost:

$$c_\phi = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\phi(i)\phi(j)} \tag{1}$$

The QAP is one of the most difficult combinatorial problems. Currently, exact algorithms are not able to solve in reasonable time instances with size $n > 30$.

An instantiation of the CGS for the QAP must define the different concepts and actions left unspecified by the CGS metaheuristic. In CGS-QAP, the ***strategy*** is implemented as a solution advertised by the consultant. It is represented by an assignment of facilities to locations, which is constructed during the

sabbatical leave. In the proposed algorithm, the sabbatical leave lasts only one iteration. In order to construct a new strategy, a consultant generates a random assignment and improves it by using a local search procedure.

The ***personal preference*** for a consultant is determined by the cost of its advertised assignment. Together with the ***reputation***, it gives the *suitability* of a consultant k:

$$suitability_k = \frac{reputation_k}{\beta + \frac{cost_k - cost_{bsf}}{cost_{bsf}}} \tag{2}$$

where the parameter β determines the influence of personal preference, $cost_k$ is the cost of the assignment advertised by consultant k and $cost_{bsf}$ is the cost of the best so far assignment. The probability to choose consultant k is:

$$p_k = \frac{suitability_k}{\sum_{c \in \mathcal{C}} suitability_c} \tag{3}$$

where $\mathcal{C}$ is the set of all available consultants. A client is allowed to choose itself as a consultant. Because the probabilities given by formula (3) do not depend on the client making the choice, the client index does not appear in this formula.

At each construction step, a client places a not yet assigned facility to a free location. In CGS-QAP, the order in which facilities are assigned to locations is random. At each step, a client receives from its consultant a recommendation regarding the location to be chosen. The recommended location is the one corresponding to the given facility in the assignment advertised by the consultant. In order to decide whether to follow the recommendation, the client uses a method inspired by the pseudorandom proportional rule introduced by the Ant Colony System [6]: with probability q_0, a client places the given facility to the location recommended by its consultant; with probability $(1 - q_0)$ it randomly places the facility to one of the free locations. The value of the parameter q_0 is critical for the performance of CGS-QAP. A large value for q_0 leads to an aggressive search, focused around the assignment advertised by the consultant. A small value for q_0 favors the exploration of the search space, allowing the algorithm to escape from local optima.

Every time a client achieves a success (i.e., it finds an assignment better than that advertised by its consultant), the consultant updates its strategy, replacing its advertised assignment with the assignment constructed by the client.

At each iteration, the consultant's k reputation fades as given by formula (4):

$$reputation_k = reputation_k(1 - r) \tag{4}$$

where the parameter r represents the reputation fading rate.

CGS-QAP prevents the reputation of a top-ranked consultant from sinking below the limit given by: $initialReputation \cdot bestSoFarCost/advertisedCost$. The value of the parameter *minReputation* referred in Fig. 3 is fixed in this algorithm at 1, because only the differences between *minReputation* and the other reputation parameters are relevant for the algorithm.

At the end of each iteration, the algorithm applies a local search procedure in order to improve the assignments constructed by clients. Since the CGS metaheuristic provides an optional local optimization step, hybridizing CGS with a local search procedure is a straightforward process. Similar to other algorithms for the QAP, CGS-QAP can use 2 opt local search, short runs of tabu search [17] or simulated annealing [5] as the local search procedure.

4.2 Experimental Setup

We run a series of experiments in order to compare the performance of CGS-QAP with that of MAX-MIN Ant System (MMAS) [15]. The choice of MMAS as a yardstick is motivated by the fact that it currently represents one of the best performing heuristics for the QAP.

To allow a meaningful comparison between heuristics, we have created a software package containing Java implementations of the algorithms considered in our experiments. The software package is available as an open source project at `http://swarmqap.sourceforge.net/`. At this address, we also provide all configuration files, problem instances and results files for the parameter tuning and for the experiments described in this paper.

Taillard [18] groups the QAP instances in four categories:

1. unstructured instances with uniform random distances and flows.
2. unstructured instances with random flows on grids.
3. structured, real-life problems.
4. structured, real-life like problems.

The instances in the first group are the most difficult to solve optimally. According to [14], in the case of MMAS the instance type has a strong influence on the local search procedure that should be used.

Our preliminary experiments have shown that this is also true for CGS-QAP: as in the case of MMAS, 2-opt local search gives better solutions for structured instances, while short runs of tabu search are preferred for unstructured instances. In this paper we concentrate on the unstructured QAP instances and we combine all algorithms with short robust tabu search runs of length $4n$, where n is the problem size.

The parameters used for MMAS are those recommended in [14]: $m = 5$ ants; $\rho = 0.8$; $\tau_{max} = \frac{1}{1-\rho}\frac{1}{J_{\psi}^{gb}}$, where J_{ψ}^{gb} is the objective function value of the global best solution; $\tau_{min} = \frac{\tau_{max}}{2 \cdot n}$. In MMAS the pheromone trails are updated using either the iteration-best solution ψ^{ib}, or the global best solution ψ^{gb}. As suggested in [14], when applying tabu search we use ψ^{gb} every second iteration.

We have tuned the parameters of CGS-QAP using the ParamILS configuration framework [9]. ParamILS executes an iterated local search in the parameter configuration space and it is appropriate for algorithms with many parameters, where a full factorial design becomes intractable. As training data for paramILS, we have used a set of 500 QAP instances with sizes uniformly distributed in the interval [30, 90]. The training instances have random distances and flows and have been generated based on the method described in [16]. The parameter settings are given in Table 1.

Table 1. Parameter settings for CGS-QAP

Parameter	**Value**	**Description**
m	10	number of virtual persons
β	0.002	influence of the advertised cost
q_0	$1 - 10/n$	probability to follow consultant's recommendation
maxReputation	40	maximum reputation value
initialReputation	15	reputation after sabbatical
bonus	6	best-so-far reputation bonus
protectedRanks	2	protected top consultants
r	0.1	reputation fading rate

4.3 Experimental Results

In our experiments we use 17 unstructured instances taken from QAPLIB [4]. For each run, we allow a total of 500 applications of tabu search. The experiments have been performed on an HP ProLiant with 8 x 2.33 GHz Intel(R) Xeon(R) CPUs and 16 GB RAM, running Red Hat Enterprise Linux 5.

Table 2 reports for each algorithm and QAP instance the mean percentage deviations from the best known solutions over 25 trials. The best mean results for each problem are in boldface. We also report for each problem the p-values

Table 2. Algorithm performance for unstructured QAP instances, averaged over 25 trials. Runs are terminated after 500 applications of tabu search.

Problem instance	Best known	MMAS (%)	CGS-QAP (%)	CGS-QAP (p-value)
tai20a	703482	0.302	**0.097**	**(0.0005)**
tai25a	1167256	0.361	**0.288**	(0.1671)
tai30a	1818146	0.436	**0.364**	(0.0441)
tai35a	2422002	0.556	**0.470**	(0.0739)
tai40a	3139370	0.719	**0.585**	(0.0122)
tai50a	4938796	1.089	**0.999**	(0.1400)
tai60a	7205962	1.257	**1.051**	**(0.0004)**
tai80a	13511780	1.380	**0.964**	**(0.0000)**
tai100a	21052466	1.420	**0.917**	**(0.0000)**
nug30	6124	0.013	**0.008**	(0.3510)
sko42	15812	0.014	**0.004**	(0.0108)
sko49	23386	0.060	**0.044**	(0.3551)
sko56	34458	0.046	**0.029**	(0.9131)
sko64	48498	0.036	**0.023**	(0.5728)
sko72	66256	0.104	**0.098**	(0.5058)
sko81	90998	0.077	**0.074**	(0.9365)
sko90	115534	**0.086**	0.120	(0.9509)

of the one-sided Wilcoxon rank sum tests for the null hypothesis (H_0) that there is no difference between the solution quality of CGS-QAP and that of MMAS, and for the alternative hypothesis (H_1) that CGS-QAP outperforms MMAS. Applying the Bonferroni correction for multiple comparisons, we obtain the adjusted α-level: $0.05/17 = 0.00294$. The p-values in boldface indicate the cases where the null hypothesis is rejected at this significance level.

Using the one-sided Wilcoxon signed rank test, we compute the p value for the null hypothesis (H_0) that there is no difference between the means of CGS-QAP and the means of MMAS, and the alternative hypothesis (H_1) that the means of CGS-QAP are smaller than the means of MMAS. The p-value obtained is 0.00019, which means that the null hypothesis can be rejected at a high significance level. Although the results are statistically significant, they do not seem to be important from a practical point of view.

In the previous experiment, CGS-QAP has outperformed MMAS on all instances except sko90. In the following experiment, we compare for this QAP instance the development of the mean percentage deviations from the best known solution for our competing algorithms over 10 trials and 10000 applications of tabu search. As shown in Fig. 5, although CGS-QAP initially produces poorer results for the sko90 instance, it is able to outperform MMAS in the long run.

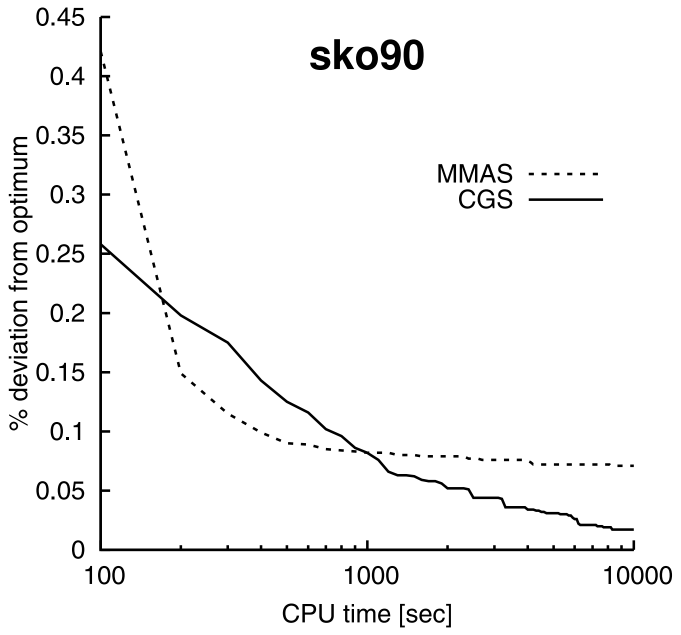

Fig. 5. Mean percentage deviations for instance sko90 averaged over 10 trials

5 Conclusions and Future Work

The experimental results show that combining CGS with a local search procedure leads to an efficient algorithm for the QAP. For unstructured QAP instances, the experimental results show that the CGS-QAP algorithm proposed in this paper performs better than MMAS. Our future research will investigate if CGS-QAP is still able to compete with MMAS for structured QAP instances.

We also plan to evaluate variants of CGS-QAP that further hybridize the metaheuristic, with the goal of improving the solution quality. One possible direction is to allow clients to ask for a second opinion. For this purpose, a client can randomly select a second consultant from the remaining ones. With a given probability, the client will choose to follow the recommendation of the second consultant instead of that of its main consultant. The solution construction bears in this case some resemblance to the approach used by genetic algorithms: choosing a location recommended by either the main or the second consultant is similar to a recombination operator; the perturbation produced when none of the recommendations is followed can be seen as a mutation.

References

1. Battiti, R., Tecchiolli, G.: The reactive tabu search. ORSA Journal on Computing 6, 126–140 (1994)
2. Birattari, M., Di Caro, G.A., Dorigo, M.: Toward the formal foundation of Ant Programming. In: Dorigo, M., Di Caro, G.A., Sampels, M. (eds.) Ant Algorithms 2002. LNCS, vol. 2463, pp. 188–201. Springer, Heidelberg (2002)
3. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, USA (1999)
4. Burkard, R.E., Karisch, S., Rendl, F.: QAPLIB - A quadratic assignment problem library. European Journal of Operational Research 55, 115–119 (1991)
5. Connolly, D.T.: An improved annealing scheme for the QAP. European Journal of Operational Research 46, 93–100 (1990)
6. Dorigo, M., Gambardella, L.M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation 1(1), 53–66 (1997)
7. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
8. Guntsch, M., Middendorf, M.: A population based approach for ACO, Applications of Evolutionary Computing. In: Cagnoni, S., Gottlieb, J., Hart, E., Middendorf, M., Raidl, G.R. (eds.) EvoIASP 2002, EvoWorkshops 2002, EvoSTIM 2002, EvoCOP 2002, and EvoPlan 2002. LNCS, vol. 2279, pp. 71–80. Springer, Heidelberg (2002)
9. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An Automatic Algorithm Configuration Framework. Journal of Artificial Intelligence Research (JAIR) 36, 267–306 (2009)
10. Iordache, S.: Consultant-Guided Search - A New Metaheuristic for Combinatorial Optimization Problems. In: Proceedings of the 2010 Genetic and Evolutionary Computation Conference (GECCO 2010). ACM Press, New York (2010)
11. Karaboga, D., Akay, B.: A Survey: Algorithms Simulating Bee Swarm Intelligence. Artificial Intelligence Review 31(1), 68–85 (2009)

12. Larrañaga, P., Lozano, J.A.: Estimation of Distribution Algorithms. In: A New Tool for Evolutionary Computation. Kluwer Academic, Boston (2001)
13. Socha, K., Dorigo, M.: Ant Colony Optimization for Continuous Domains. European Journal of Operational Research 185(3), 1155–1173 (2008)
14. Stützle, T., Dorigo, M.: ACO Algorithms for the Quadratic Assignment Problem. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 33–50. McGraw-Hill, New York (1999)
15. Stützle, T., Hoos, H.H.: MAX-MIN Ant System. Future Generation Computer Systems 16(8), 889–914 (2000)
16. Stützle, T., Fernandes, S.: New Benchmark Instances for the QAP and the Experimental Analysis of Algorithms. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2004. LNCS, vol. 3004, pp. 199–209. Springer, Heidelberg (2004)
17. Taillard, E.D.: Robust taboo search for the quadratic assignment problem. Parallel Computing 17, 443–455 (1991)
18. Taillard, E.D.: Comparison of iterative searches for the quadratic assignment problem. Location Science 3, 87–105 (1995)
19. Zlochin, M., Birattari, M., Meuleau, N., Dorigo, M.: Model-based search for combinatorial optimization: A critical survey. Annals of Operations Research 131, 373–395 (2004)