Ant Colonies for the QAP

L. M. GAMBARDELLA, É. D. TAILLARD
IDSIA, Corso Elvezia 36, CH-6900 Lugano, Switzerland
luca@idsia.ch, eric@idsia.ch, http://www.idsia.ch

M. DORIGO
IRIDIA - CP 194/6, Université Libre de Bruxelles
Avenue Franklin Roosevelt 50, 1050 Bruxelles, Belgium
mdorigo@ulb.ac.be, http://iridia.ulb.ac.be/dorigo/dorigo.html
tel: +32-2-6503169, fax: +32-2-6502715

TECHNICAL REPORT, IDSIA, 4, 1997

This paper presents HAS-QAP, an hybrid ant colony system coupled with a local search, applied to the quadratic assignment problem. HAS-QAP uses trail information to perform modifications on QAP solutions in contrast with more traditional ant system that uses trail information to construct complete solutions. HAS-QAP is analysed and compared with some of the best heuristics available for the QAP: two taboo search versions, that is, robust and reactive taboo search, an hybrid genetic algorithm, and a simulated annealing. Experimental results show that HAS-QAP and the hybrid genetic algorithm are the best performing on real world, irregular and structured problems due to their ability in finding the structure of good solutions, while HAS-QAP performance is less competitive on random, regular and unstructured problems.

Key words: Quadratic assignment problem, ant colony optimisation, ant systems, meta-heuristics.

INTRODUCTION

The quadratic assignment problem

The QAP is a combinatorial optimisation problem stated for the first time by Koopmans and Beckman[1] in 1957. It can be described as follows: Given two $n \times n$ matrices A=(a_{ij}) and B=(b_{ij}), find a permutation π^* minimising

$$\min_{\pi \in \Pi(n)} f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot b_{\pi_i \pi_j}$$

where $\Pi(n)$ is the set of permutations of n elements. Shani and Gonzalez[2] have shown that the problem is *NP*-hard and that there is no ε-approximation algorithm for the QAP unless $P = NP$. While few combinatorial optimisation problems can be solved exactly for relatively large instances, as exemplified by the travelling salesman problem (TSP), QAP instances of size larger than 20 are considered intractable. In practice, a large number of real world problems lead to QAP instances of considerable size that cannot be solved exactly. For example, an application in image processing requires to solve more than 100 problems of size $n = 256$ (Taillard[3]). The use of heuristic methods for solving large QAP instances is currently the only practicable solution.

Heuristic methods for the QAP

A large number of heuristic methods have been developed for solving the QAP, using various techniques. Very shortly, the most remarkable methods are the following: the

simulated annealing of Connolly[4], the taboo searches of Taillard[5], Battiti and Tecchiolli[6] and Sondergeld and Voß[7], and the hybrid genetic-taboo search of Fleurent and Ferland[8]. More recently, another promising method has been developed based on scatter search by Cung, Mautor, Michelon and Tavares[9]. This new method embeds a taboo search.

This paper presents an hybrid ant-local search system for the QAP that works often better than the above mentioned heuristics on structured problems. The paper is organised as follows: In the next section we briefly describe ant systems in general. The following section presents our hybrid ant-local search system. We then analyse the performance of our algorithm by comparing it to the best existing methods, and in the last section we draw some conclusions.

ANT SYSTEM

The idea of imitating the behaviour of ants for finding good solutions to combinatorial optimisation problems was initiated by Dorigo, Maniezzo and Colorni[10,11], and Dorigo[12]. The principle of these methods is based on the way ants search for food and find their way back to the nest[13]. Initially, ants explore the area surrounding their nest in a random manner. As soon as a source of food is found by an ant, the last evaluates the interest of the source (quantity and quality) and carries some of this food to the nest. During the return trip, the ant leaves a chemical pheromone trail (pheromone) on the ground. The rôle of this pheromone trail is to guide other ants toward the source of food and the quantity of pheromone left by an ant depends on the amount of food found. After a while, the path to the food source will be indicated by a strong pheromone trail and the more the ants which reach the source of food, the stronger the pheromone trail left is.

Since sources that are close to the nest are visited more frequently than those that are far away, pheromone trails leading to the nearest sources grow more rapidly. These pheromone trails are exploited by ants as a means to find their way from nest to food source and back.
The transposition of real ants food searching behaviour into an algorithmic framework for solving combinatorial optimisation problems is done by making an analogy between 1) the real ants search area and the set of feasible solutions to the combinatorial problem, 2) the amount of food in a source and the objective function, and 3) the pheromone trail and an adaptive memory. A detailed description of how these analogies can be put to work in the TSP case can be found in Dorigo, Maniezzo and Colorni[10]. An up-to-date list of papers and applications of ant colony optimisation algorithms is maintained on the WWW at the address: http://iridia.ulb.ac.be/dorigo/ACO/ACO.html.

The present paper shows how these analogies can be put to work in the case of the quadratic assignment problem (QAP). The resulting method, Hybrid Ant System for the QAP, HAS-QAP for short, is shown to be efficient for solving structured QAPS.

HYBRID ANT SYSTEM FOR THE QAP

In this section we present the HAS-QAP basic algorithm and we compare and analyse its main components in relation to previous ant inspired systems. Very shortly, Hybrid Ant System for the QAP is based on the schematic algorithm of Figure 1:

FIGURE 1. *The HAS-QAP algorithm structure.*

```
Generate m initial solutions, each one associated to one ant
Initialise the pheromone trail
    For I^max iterations repeat:
        For each ant k = 1, ..., m do:
            Modify the solution associated to ant k using the pheromone
                trail
            Apply a local search to the modified solution
            Associate a new starting solution to ant k using an
                intensification mechanism
        Update the pheromone trail
        Apply a diversification mechanism
```

Pheromone trail

The most important component of an ant system is the management of pheromone trails. In a standard ant system, pheromone trails are used in conjunction with the objective function for constructing a new solution. Informally, pheromone levels give a measure of how desirable is to insert a given element into a solution. For example, in Dorigo and Gambardella's[14] TSP application, links get a higher amount of pheromone if they are used by ants which make shorter tours. Pheromone trails in this case are maintained in a pheromone matrix $\boldsymbol{T} = (\tau_{ij})$ where τ_{ij} measures how desirable is the edge between cities *i* and *j*. For the QAP, we have also chosen to represent the set of the pheromone trails by a matrix $\boldsymbol{T} = (\tau_{ij})$ of size $n \times n$, where the entry τ_{ij} measures the desirability of setting $\pi_i = j$ in the solutions visited by the ant system.

There are two different uses of pheromone: *exploration* and *exploitation*. Exploration is a stochastic process in which the choice of the component used to construct a solution to the problem is chosen in a probabilistic way. Exploitation chooses the component that maximises a blend of pheromone trail values and partial objective function evaluations.

After having built a new solution, a standard ant system updates the pheromone trails as follows: first, all the pheromone trails are decreased to simulate the evaporation of pheromone. Then, the pheromone trails corresponding to components that were chosen for constructing the solution are reinforced, taking into consideration the quality of the solution.

Since the first implementation of an ant system (Dorigo, Maniezzo and Colorni[10]), it has been changed in many ways so to improve its efficiency. In our best ant system to date (Dorigo and Gambardella[14]) pheromone trails update is very loosely coupled with what happens with real ants. Indeed, pheromone trails are not only modified directly and *locally* by the artificial agents during or just after the construction of a new solution, but also *globally*,

considering the best solution generated by all the agents at a given iteration or even the best solution ever constructed. In HAS-QAP, we do not perform a local update of the pheromone trails, but only a global update. This makes the search more aggressive and requires less time to reach good solutions. Moreover, this has been strengthened by an *intensification* mechanism. An inconvenience of such a process is the risk of an early convergence of the algorithm. Therefore, we design a *diversification* mechanism that periodically erases all the pheromone trails.

Solutions manipulation

Another peculiarity of HAS-QAP is the use of the pheromone trails in a non standard way: in preceding applications of ant systems (Dorigo, Maniezzo and Colorni[10,11], Dorigo and Gambardella[14]), pheromone trails were exploited to build a completely new solution. Here we use pheromone trails to *modify* an existing solution, in the spirit of a neighbourhood search.

After an artificial agent has modified a solution, taking into account the information contained in the pheromone trail matrix only, we apply an improvement phase that consists in performing a fast local search that takes into consideration only the objective function. Adding local search to ant systems has also been identified as very promising by other researchers: for the TSP, Dorigo and Gambardella[14] succeeded in designing an ant system almost as efficient as the best implementations of the Lin and Kernighan heuristic by adding a simple 3-opt phase after the construction phase and for the sequential ordering problem (SOP) Gambardella and Dorigo[15] alternate an ant system with a local search specifically designed for the SOP. In fact, many of the most efficient heuristic methods for combinatorial optimisation problems are based on a neighbourhood search, either a greedy local search or a more elaborate meta-heuristic like taboo search (e.g., see Battiti and Tecchiolli[6], Cung et al.[9], Fleurent and Ferland[8], Taillard[5], Sondergeld and Voß[7] for applications to the QAP) or simulated annealing (see e. g. Burkard and Rendl[16], and Connolly[4]).

Intensification and diversification

As we said, in HAS-QAP each ant has associated a problem solution that is first modified using pheromone trail and later on is improved using a local search mechanism. This mechanism is completely different from any other previous ants system where solutions were not explicitly associated to ants but were created in each iteration using a constructive mechanism.

Therefore, in HAS-QAP, we deal with the problem of choosing, iteration after iteration, the starting solution associated to each ant. To solve this problem we have defined two mechanisms called *intensification* and *diversification*. Intensification is used to explore the neighbourhood of good solutions more completely: when intensification is active, the ant comes back to the solution it has at the beginning of the iteration if this solution is better than the solution it has at the end of the iteration; in all other cases, the ant simply continues working ont its current solution. Diversification implements a partial restart of the algorithm

when the solutions seems not improving any more. It consists in a re-initialisation of both the pheromone trail matrix and solutions associated to ants.

HAS-QAP DESCRIPTION

In this section we discuss the most salient aspects of our HAS-QAP algorithm, which is presented into details in Figure 2.

FIGURE 2. *The HAS-QAP algorithm.*

```
/* initialisation */
Generate m random initial permutations π¹(1), …, πᵐ(1), each one
associated to an ant
Improve π¹(1), …, πᵐ(1) with the local search procedure
Let π* be the best solution
Initialise the pheromone trail matrix T
Activate intensification
/* main loop */
For i = 1 to I^max repeat:
    /* solution manipulation */
    For each permutation π^k(i) (1 ≤ k ≤ m) do:
        Apply R pheromone trail swaps to π^k(i) to obtain π̂^k(i)
        Apply the local search procedure to π̂^k(i) to obtain π̃^k(i)
    /* intensification */
    For each ant k do:
        If intensification is active then
            π^k(i+1) ← best permutation between π^k(i) and π̃^k(i)
        Else
            π^k(i+1) ← π̃^k(i)
    If π^k(i+1)= π^k(i) ∀k then deactivate intensification
    If ∃k such that f(π̃^k(i)) < f(π*) then
        Update π*, the best solution found so far
        Activate intensification
    /* pheromone trial updating */
    Update the pheromone trail matrix
    /* diversification */
    If S iterations have been performed without improving π* then
        Perform a diversification
```

Solutions Initialisation

Initially, each ant is given a randomly chosen permutation. These permutations are initially optimised using the same local search procedure that will be described further.

Pheromone trail initialisation

Initially no information is contained in the pheromone trail matrix, meaning that all pheromone trails τ_{ij} are equal to a value τ_0. Since pheromone trails are updated by taking into account the absolute value of the solution obtained, τ_0 must take a value that depends on the value of the solutions that will be visited. Therefore, we have chosen to set $\tau_0 = 1/(Q \cdot f(\pi^*))$,

where π^* is the best solution found so far and Q a parameter. The re-initialisations of the pheromone trail are done in the same way, but π^* might have changed.

Pheromone trail based modification

Pheromone trail based modification is applied by each ant to its own permutation π^k. It starts with π^k and produces a permutation $\hat{\pi}^k$. It consists in repeating R of the following swaps. First, an index r is chosen, randomly between 1 and n. Then, a second index $s \neq r$ is chosen and the elements π_r^k and π_s^k are swapped in the current solution π^k. The second index is chosen according to one of two different randomly chosen policies. With a probability given by the parameter q, s is chosen in such a way that $\tau_{r\pi_s}^k + \tau_{s\pi_r}^k$ is maximum. This policy consists in *exploiting* the pheromone trail. The second policy, chosen with probability (1-q), consists in *exploring* the solution space by choosing the second index s with a probability proportional to the values contained in the pheromone trail. More precisely, s is chosen with probability:

$$\frac{\tau_{r\pi_s}^k + \tau_{s\pi_r}^k}{\sum_{j \neq r} (\tau_{r\pi_j}^k + \tau_{j\pi_r}^k)}$$

Local search

Local search consists in applying twice a complete neighbourhood examination with first improving strategy to a solution $\hat{\pi}^k$ to produce a solution $\tilde{\pi}^k$. The local search procedure is shown in Figure 3, where $\Delta(\pi, i, j)$ is the difference in the objective function value when exchanging the elements π_i and π_j in π. The evaluation of $\Delta(\pi, i, j)$ can be performed in $O(n)$ using the following formula:

$$\Delta(\pi, i, j) = (a_{ii} - a_{jj})(b_{\pi_j\pi_j} - b_{\pi_i\pi_i}) + (a_{ij} - a_{ji})(b_{\pi_j\pi_i} - b_{\pi_i\pi_j}) + \sum_{k \neq i,\, j} (a_{ki} - a_{kj})(b_{\pi_k\pi_j} - b_{\pi_k\pi_i}) + +(a_{ik} - a_{jk})(b_{\pi_j\pi_k} - b_{\pi_i\pi_k})$$

FIGURE 3. *The complete neighbourhood examination procedure with first improving strategy procedure.*

```
I = ∅.
While |I| < n repeat:
    Choose i, uniformly random, 1 ≤ i ≤ n, i ∉ I.
    J = {i}
    While |J| < n repeat:
        Choose j, uniformly random, 1 ≤ j ≤ n, j ∉ J.
        If Δ(π, i, j) < 0, exchange π_i and π_j in π.
        J = J ∪ {j}
    I = I ∪ {i}.
```

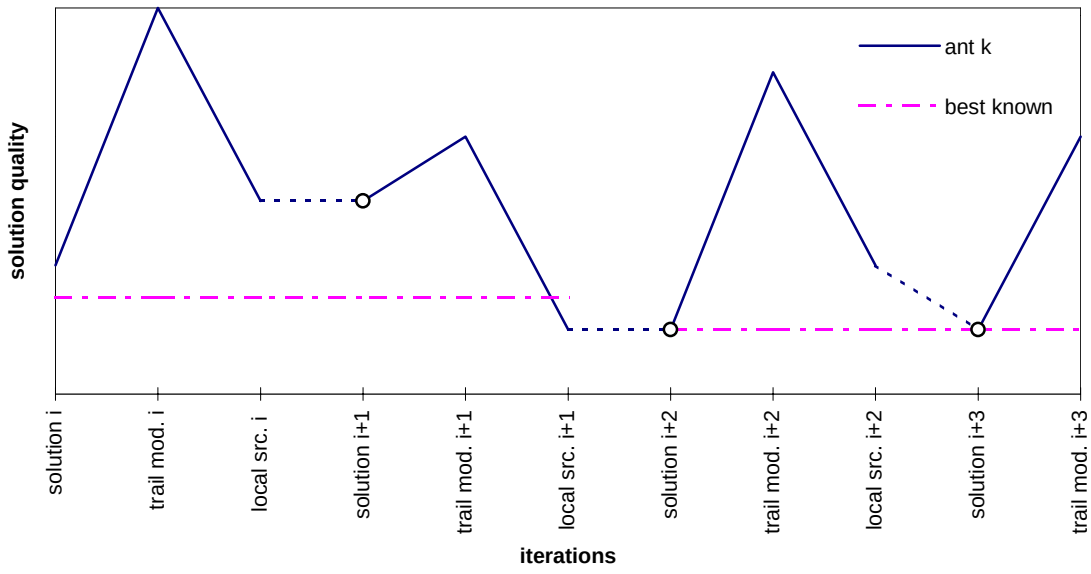
This procedure examines systematically all the possible swaps and performs immediately an improving swap, if one is found. The order in which the swaps are examined is randomly chosen. Therefore the local search procedure examines $O(n^2)$ swaps and can be executed in $O(n^3)$; it does not necessarily reach a local optimum, but is fast and may produce different

solutions when starting with the same initial, not locally optimal solution. To speed-up the local search, it is clear that it is not necessary to compute $\Delta(\pi, i, j)$ if $i = j$ and the second neighbourhood examination is useless if the first one did not find an improving move.

Intensification

The function of intensification is to explore the neighbourhood of good solutions more completely. The intensification mechanism (see Figure 4) is activated when the best solution produced by the search so far has been improved. When intensification is active each ant starts its next iteration with the best permutation between π^k and $\tilde{\pi}^k$. This is different from what happens when intensification is not active, in which case the permutation maintained is $\tilde{\pi}^k$. The intensification flag remains active while at least one ant succeeds in improving its solution during an iteration. The rationale for intensification is that it favours search in the neighbourhood of the new best solution found. In fact, since trail updating is governed by the value π^* at a given iteration, the trail distribution is determined by past values of π^*. When a new solution π^{k*} is found such that $\pi^{k*} < \pi^*$, π^{k*} becomes the new π^*. In general it will take some iterations before the influence of the new π^* on trail distribution will have all its effect. Intensification focuses search around new π^* while information about new π^* grows into the pheromone trail matrix $\boldsymbol{T}$. In Figure 4 we show the behaviour of a generic ant k according to intensification mechanism: on the horizontal axis we report, iteration after iteration, three search steps: initial solution, pheromone modification and local search. On the vertical axis we measure the value of these permutations. At the beginning of iteration i, let us suppose that intensification is not active therefore $\pi^k(i+1) \leftarrow \tilde{\pi}^k(i)$. At the end of iteration $i+1$, $\pi^k(i+2)$ is set again to $\tilde{\pi}^k(i+1)$ and intensification is activated because the best known solution is improved. At the end of iteration $i+2$, $\pi^k(i+3)$ is set to $\pi^k(i+2)$ because intensification is active and $\pi^k(i+2)$ is a better solution than $\tilde{\pi}^k(i+2)$.

FIGURE 4. *Intensification mechanism.*



Pheromone trail update

The update of the pheromone trail is done in a different way than that of the standard ant system where all the ants update the pheromone trail with the result of their computation. Indeed, this manner of updating the pheromone trail implies a very slow convergence of the algorithm (Dorigo and Gambardella[14]). For speeding-up the convergence, we update the pheromone trail by taking into account only the best solution produced by the search to date. First, all the pheromone trails are weakened by setting $\tau_{ij} = (1 - \alpha_1)\cdot\tau_{ij}$, $(1 \le i, j \le n)$ where $0 < \alpha_1 < 1$ is a parameter that controls the *evaporation* of the pheromone trail: a value of α_1 close to 0 implies that the pheromone trails remain active a long time, while a value close to 1 implies a high degree of evaporation and a shorter memory of the system. Then, the pheromone trail is reinforced by considering only $\boldsymbol{\pi}^*$, that is, the best solution generated by the system so far and setting $\tau_{i\pi_i^*} = \tau_{i\pi_i^*} + \alpha_2/f(\boldsymbol{\pi}^*)$ for all i.

Diversification

The aim of diversification is to constrain the algorithm to work on solutions with different structure. The diversification mechanism is activated if during the last S iterations no improvement to the best generated solution is detected. Diversification consists in erasing all the information contained in the pheromone trail by a re-initialisation of the pheromone trail matrix and in generating randomly a new current solution for all the ants but one that receives the best solution produced by the search so far.

Complexity and memory requirement.

The complexity of our approach can be evaluated as follows: the most time consuming part of the algorithm is the local search procedure. The complexity of this step is $O(n^3)$ and it is repeated $I^{max}m$ times. Therefore the total complexity of HAS-QAP is $O(I^{max}mn^3)$. The memory size requied by the algorithm is $O(n^2 + nm)$ since the data matrices have to be stored as well as m permutations.

NUMERICAL RESULTS

We compare HAS-QAP with a number of the best heuristic methods available for the QAP. For the comparison, we consider a large subset of well known problem instances, having sizes between $n = 19$ and $n = 90$, contained in the QAPLIB compiled by Burkard, Karisch and Rendl[17].

As shown by Taillard[3], the quality of solutions produced by heuristic methods strongly depends on the problem type, that is, on the structure of the data matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. Generally, instances of problems taken from the real world present at least one matrix with very variable entries, for example with a majority of 0. For such problems, many heuristic methods perform rather poorly, being unable to find solutions below 10% above the best solutions known even if an excessive computing time is allowed. Moreover, this occurs also for problems of small

size. Conversely, the same methods may perform very well on problems randomly generated with matrices having entries uniformly distributed. For such problems, almost all heuristic methods are able to find high quality solutions (i.e., solutions approximately one per cent worse than the best solution known).

For real world problems, Taillard[3] found that the genetic hybrid method of Fleurent and Ferland[8] was one of the best performing, in particular in terms of the quality of the solutions obtained. For problems with matrices having entries uniformly distributed, the reactive taboo search of Battiti and Tecchiolli[6], and the taboo search of Taillard[5] perform best. Therefore, we analyse the performances of HAS-QAP by splitting the problem instances into two categories: (i) real world, irregular and structured problems, and (ii) randomly generated, regular and unstructured problems. We have classified the problems using a simple statistics known as "flow-dominance"[18, 19]. Let μ_A (respectively μ_B) be the average value of the off-diagonal elements of matrix $\boldsymbol{A}$ (respectively $\boldsymbol{B}$) and σ_A (respectively σ_B) the corresponding standard deviation. A problem is put in category (i) if $\max(\sigma_A/\mu_A, \sigma_B/\mu_B) > 1.2$ and in category (ii) otherwise. In tables 1 and 3 we report the flow dominance value for the tested problems.

All the algorithms used for comparisons were run with the parameter settings proposed by their authors, except for GH, where we have used a population of min(100, 2n) solutions instead of 100 solutions in order to improve the method (see also Taillard[3]). It is important to note that RTS is a method with self-adapting parameters (i.e., the only parameter set by the user is the number of iterations) and the SA cooling scheme is also self-adapted, with the total number of iterations set by the user.

The number of iterations performed with the different methods were limited to have similar computation times. In addition, we provide results for two values of I^{max}: short runs with I^{max}=10 and long runs with I^{max}=100. The reason to compare algorithms on short and on long runs is to evaluate their ability in producing relatively good solutions in short time versus producing very good solutions in longer runs. In addition, short runs are interesting in case these methods are used to produce starting solutions for different algorithms or to deal with problems of large size.

HAS-QAP parameters setting

For HAS-QAP we used the following parameter settings: $\alpha_1 = \alpha_2 = 0.1$, $m = 10$, $Q = 100$, $R = n/3$, $S = n/2$ and $q = 0.9$. These parameters were experimentally found to be good and robust for the problems tested.

In particular, if a fixed and long computation effort is allowed, a number of ants between 8 to 15 always produces better solutions than running the system with only one ant. In fact, with a single ants the system capability to escape from local minima decreases. A number of ants greater than 20 has been tested to be not efficient because system ability to escape from local minima does not increase a lot while the quality of the solutions generated decreases due to inferior number of iterations executed. On the contrary, for short runs a single ant might be

desirable because more trail updating and more local searches are performed in the same amount of time.

For parameter R, the number of swap exchanges executed using trail information, $R = n/3$ has been experimentally tested to be more efficient than $n/2$ and $n/6$. Starting from π^k we generated a new permutation $\hat{\pi}^k$ by performing R trail driven exchanges and we optimise $\hat{\pi}^k$ producing permutation $\tilde{\pi}^k$ using the local search procedure. In case of $R \geq n/2$ the resulting permutation $\hat{\pi}^k$ tends to be too close to solution π^* used to perform global trial updating and improving solutions are difficult to be computed. On the contrary, $R \leq n/6$ does not allow the system to escape from local minima because, after the local search, $\tilde{\pi}^k$ was in most situations the same as the starting permutation π^k.

Same type of experiments and considerations has been made to define other parameters, like $\alpha_1=\alpha_2$ and Q=100. In addition, we tested the importance of diversification (parameter S) and exploration (parameter q) by running different experiments where $S = \infty$ (no diversification) and/or $q = 1$ (no exploration) were allowed. In all these situations algorithm performance was worse in comparison with the performance obtained using the proposed values ($S = n/2$ and $q = 0.9$). For S close to n and $S \leq n/6$ phenomena of stagnation and insufficient intensification have been observed.

In general we have experimentally tested that the proposed parameter setting is very robust in relation to slight modification of the proposed values.

Real world, irregular and structured problems

From the QAPLIB, we have selected 20 problems having values of the flow-dominance statistic larger than 1.2. Most of these problems are issued from practical applications or have been randomly generated with non uniform laws, imitating the distributions observed on real world problems. As mentioned above, we consider for our comparisons the taboo searches of Battiti and Tecchiolli[6] (RTS) and Taillard[5] (TT), and the genetic-hybrid method of Fleurent and Ferland[8] (GH). We have also considered a simulated annealing due to Connolly[4] (SA) that is cited as a good implementation by Burkard and Çela[20]. We have re-programmed the GH, TT and SA methods, and we have used the original code of RTS. Unfortunately, this code only considers symmetric problems. Therefore, problems with an asymmetric matrix have not been solved with RTS.

Table 1 compares all these methods on short executions: $I^{max} = 10$ for HAS-QAP. Allowing about the same computing time as the one used by HAS-QAP (or a slightly higher computing time), it is possible to perform $100n$ iterations of RTS and TT, $1250n^2$ iterations of SA and 25 iteration of GH. In Table 1, we provide the average quality of the solutions produced by these methods, measured in per cent above the best solution value known; the last column of this table reports the computing time (seconds) needed by HAS-QAP for performing 10 iterations (seconds on Sun Sparc 5; all the other algorithms were allocated the same or slightly more computing time). All the methods were run 10 times. Best results are in italic boldface. To assess the statistical significance of the differences among means we have

run the Mann-Whitney U-test (this is the non-parametric counterpart of the *t*-test for independent samples; the *t*-test cannot be used here because of the limited dimensions of the samples and for the nature of data). For each problem, that is for each row of the table, we run the Mann-Whitney U-test between the best result (italic boldface) and each of the results obtained with the other meta-heuristics. Whenever the two results are not significantly different at the 0.1 level, this is indicated in the table by boldface characters (e.g., in Table 1 for problem bur26b, GH resulted to be the best meta-heuristic; SA and HAS-QAP were found to be not statistically different at the 0.1 level; we can therefore say that for this problem SA, GH, and HAS-QAP were the best meta-heuristics). The same procedure was applied in the following tables.

TABLE 1. *Quality of various heuristic methods for irregular problems and short runs. Best results are in boldface.*

Problem name	flow dom.	*n*	Best known value	TT $100n$	RTS $100n$	SA $1250n^2$	GH 25	HAS-QAP 10	HAS-QAP 10 CPU
bur26a	2.75	26	5426670	0.208	—	0.185	0.060	***0.027***	5.2
bur26b	2.75	26	3817852	0.441	—	**0.191**	***0.090***	**0.106**	5.1
bur26c	2.29	26	5426795	0.170	—	0.137	***0.004***	**0.009**	5.2
bur26d	2.29	26	3821225	0.249	—	0.379	**0.003**	***0.002***	5.0
bur26e	2.55	26	5386879	0.076	—	0.228	***0.003***	**0.004**	5.1
bur26f	2.55	26	3782044	0.369	—	0.224	0.006	***0.000***	5.0
bur26g	2.84	26	10117172	0.078	—	0.139	0.006	***0.000***	5.1
bur26h	2.84	26	7098658	0.349	—	0.368	**0.003**	***0.001***	5.0
chr25a	4.15	25	3796	**15.969**	16.844	27.139	***15.158***	**15.690**	4.1
els19	5.16	19	17212548	21.261	6.714	16.028	***0.515***	**0.923**	2.3
kra30a	1.46	30	88900	2.666	2.155	**1.813**	***1.576***	**1.664**	8.0
kra30b	1.46	30	91420	**0.478**	1.061	1.065	***0.451***	**0.504**	9.0
tai20b	3.24	20	122455319	6.700	—	14.392	***0.150***	**0.243**	2.7
tai25b	3.03	25	344355646	11.486	—	8.831	0.874	***0.133***	5.2
tai30b	3.18	30	637117113	13.284	—	13.515	0.952	***0.260***	9.3
tai35b	3.05	35	283315445	10.165	—	6.935	1.084	***0.343***	15.5
tai40b	3.13	40	637250948	9.612	—	5.430	1.621	***0.280***	24.4
tai50b	3.10	50	458821517	7.602	—	4.351	1.397	***0.291***	51.2
tai60b	3.15	60	608215054	8.692	—	3.678	2.005	***0.313***	91.5
tai80b	3.21	80	818415043	6.008	—	2.793	2.643	***1.108***	224.5

From Table 1, it is clear that methods like TT or SA are not well adapted for irregular problems. Sometimes, they produce solutions more than 10% above the best solutions known while other heuristic methods are able to exhibit solutions at less than 1% with the same computing effort. For the problem types *bur...* and *tai..b*, our HAS-QAP seems to be the best method overall. The only real competitor of HAS-QAP for irregular problems is GH. (Or, more precisely, the repetition of 25 independent taboo searches with 4*n* iterations: Indeed, the genetic algorithm of Fleurent and Ferland[8] is hybridised with TT; each solution produced by this method is improved with 4*n* iterations of TT before being eventually inserted in the population. After 25 iterations, the GH algorithm is still in the initialisation phase that consists in building initial solutions issued from min(100, 2*n*) repetitions of TT starting with random initial solutions). SA is never really competitive. This is surprising since this method was allowed to perform a much higher number of iterations than the other methods, but the

author of SA already mentioned that his method performs less well on problems with high flow dominance.[4]

In Table 2, we compare our HAS-QAP on longer runs, with I^{max} = 100, 1000*n* iterations of RTS and TT, 12500n^2 iterations of SA, and 250 iterations of GH. When all the 10 runs of HAS-QAP succeeded in finding the best solution known, we put in parentheses the average computing time (seconds) for reaching the best solution known; the time for completing 100 iterations is roughly 10 times that for 10 iterations. From this table, we draw the same conclusions as for shorter runs. As above, we ran Mann-Whitney U-tests and found that for all instances but two HAS-QAP resulted to be one of the best methods.

TABLE *2. Quality of various heuristic methods for irregular problems and long runs. Best results are in boldface.*

Problem name	*n*	Best known value	TT 1000*n*	RTS 1000*n*	SA 12500n^2	GH 250	HAS-QAP 100	HAS-QAP 100 CPU
bur26a	26	5426670	0.0004	—	0.1411	0.0120	***0***	(10)
bur26b	26	3817852	0.0032	—	0.1828	0.0219	***0***	(17)
bur26c	26	5426795	0.0004	—	0.0742	***0***	***0***	(3.7)
bur26d	26	3821225	0.0015	—	0.0056	0.0002	***0***	(7.9)
bur26e	26	5386879	***0***	—	0.1238	***0***	***0***	(9.1)
bur26f	26	3782044	0.0007	—	0.1579	***0***	***0***	(3.4)
bur26g	26	10117172	0.0003	—	0.1688	***0***	***0***	(7.7)
bur26h	26	7098658	0.0027	—	0.1268	0.0003	***0***	(4.1)
chr25a	25	3796	6.9652	9.8894	12.4973	***2.6923***	**3.0822**	40
els19	19	17212548	***0***	0.0899	18.5385	***0***	***0***	(1.6)
kra30a	30	88900	0.4702	2.0079	1.4657	***0.1338***	0.6299	76
kra30b	30	91420	**0.0591**	0.7121	0.1947	***0.0536***	**0.0711**	86
tai20b	20	122455319	***0***	—	6.7298	***0***	0.0905	27
tai25b	25	344355646	0.0072	—	1.1215	***0***	***0***	(12)
tai30b	30	637117113	0.0547	—	4.4075	0.0003	***0***	(25)
tai35b	35	283315445	0.1777	—	3.1746	0.1067	***0.0256***	147
tai40b	40	637250948	0.2082	—	4.5646	0.2109	***0***	(51)
tai50b	50	458821517	**0.2943**	—	0.8107	**0.2142**	***0.1916***	480
tai60b	60	608215054	0.3904	—	2.1373	0.2905	***0.0483***	855
tai80b	80	818415043	1.4354	—	1.4386	**0.8286**	***0.6670***	2073

Randomly generated, regular and unstructured problems

For unstructured problems, the solutions produced by HAS-QAP are not as good as for structured problems. This can be explained by the fact that relatively good solutions of these problems are spread in the whole feasible solution set (see the entropy measure of local optima reported in Taillard[3]).

The main strength of HAS-QAP, as well as of the GH algorithm, is its ability in finding the structure of good solutions. If all the good solutions are concentrated in a close subset of the feasible solutions, GH and HAS-QAP work well. On the contrary, these algorithms are not performing well when a large number of relatively good solutions are spread all over the solutions space. In tables 3 and 4, we provide the same results as for tables 1 and 2, but for unstructured problems.

From these tables, we can conclude that TT, GH and SA performs best for *sko...* problems and RTS performs best for *tai..a* problems while our HAS-QAP is not so good and even the worst method for *tai..a* problems.

TABLE 3. *Quality of various heuristic methods for regular problems and short runs. Best results are in boldface.*

Problem name	flow dom.	*n*	Best known value	TT 100*n*	RTS 100*n*	SA $1250n^2$	GH 25	HAS-QAP 10	HAS-QAP 10 CPU
nug20	0.99	20	2570	**0.101**	0.911	0.327	***0.047***	**0.156**	2.5
nug30	1.09	30	6124	**0.271**	0.872	0.500	***0.249***	0.565	8.7
sko42	1.06	42	15812	***0.187***	1.116	0.301	0.477	0.654	26
sko49	1.07	49	23386	***0.198***	0.978	0.406	0.368	0.661	45
sko56	1.09	56	34458	***0.347***	1.082	**0.504**	**0.515**	0.729	69
sko64	1.07	64	48498	***0.221***	0.861	**0.390**	0.631	0.504	105
sko72	1.06	72	66256	**0.478**	0.948	***0.323***	0.616	0.702	153
sko81	1.05	81	90998	**0.304**	0.880	***0.289***	0.628	0.493	222
sko90	1.06	90	115534	***0.386***	0.748	**0.418**	0.632	0.591	307
tai20a	0.61	20	703482	**0.769**	***0.705***	1.209	0.732	1.483	2.7
tai25a	0.60	25	1167256	**1.128**	***0.892***	1.766	1.371	2.527	5.1
tai30a	0.59	30	1818146	***0.871***	**1.044**	1.434	**1.160**	2.600	9.2
tai35a	0.58	35	2422002	**1.356**	***1.192***	1.886	1.455	2.969	15
tai40a	0.60	40	3139370	1.284	***0.996***	1.750	1.590	3.063	24
tai50a	0.60	50	4941410	**1.377**	***1.241***	2.296	1.841	3.487	50
tai60a	0.60	60	7208572	1.544	***1.248***	1.942	1.867	3.686	88
tai80a	0.59	80	13557864	1.170	***0.749***	1.773	1.344	2.996	220
wil50	0.64	50	48816	***0.137***	0.504	**0.149**	0.253	**0.211**	47

TABLE 4. *Quality of various heuristic methods for regular problems and long runs. Best results are in boldface.*

Problem name	*n*	Best known value	TT 1000*n*	RTS 1000*n*	SA $12500n^2$	GH 250	HAS-QAP 100	HAS-QAP 100 CPU
nug20	20	2570	***0***	0.911	0.070	***0***	***0***	(3.6)
nug30	30	6124	0.032	0.872	0.121	***0.007***	0.098	83
sko42	42	15812	0.039	1.116	0.114	***0.003***	0.076	248
sko49	49	23386	**0.062**	0.978	0.133	***0.040***	0.141	415
sko56	56	34458	**0.080**	1.082	**0.110**	***0.060***	**0.101**	639
sko64	64	48498	***0.064***	0.861	**0.095**	***0.092***	**0.129**	974
sko72	72	66256	**0.148**	0.948	**0.178**	***0.143***	0.277	1415
sko81	81	90998	***0.098***	0.880	**0.206**	0.136	**0.144**	2041
sko90	90	115534	***0.169***	0.748	**0.227**	**0.196**	**0.231**	2825
tai20a	20	703482	***0.211***	**0.246**	0.716	**0.268**	0.675	26
tai25a	25	1167256	**0.510**	***0.345***	1.002	0.629	1.189	50
tai30a	30	1818146	**0.340**	***0.286***	0.907	0.439	1.311	87
tai35a	35	2422002	0.757	***0.355***	1.345	0.698	1.762	145
tai40a	40	3139370	1.006	***0.623***	1.307	0.884	1.989	224
tai50a	50	4941410	1.145	***0.834***	1.539	1.049	2.800	467
tai60a	60	7208572	1.270	***0.831***	1.395	1.159	3.070	820
tai80a	80	13557864	0.854	***0.467***	0.995	0.796	2.689	2045
wil50	50	48816	**0.041**	0.504	0.061	***0.032***	0.061	441

CONCLUSIONS

Recent results in the application of heuristic methods to the solution of difficult combinatorial problems seem to indicate that combining local optimisation with good heuristic ways of generating starting solutions for the local searches can give raise to powerful, although approximate, optimisation algorithms. Ant colony optimisation is a recent heuristic loosely based on the observation of some aspects of ant colonies behaviour (Dorigo, Maniezzo and Colorni[10,11], Dorigo[12]). In this paper we have presented results on the quadratic

assignment problem obtained by HAS-QAP, an algorithm which interleaves an ant colony algorithm with a simple local search. Comparisons with some of the best heuristics for the QAP have shown that our approach is among the best as far as real world, irregular, and structured problems are concerned. The only competitor resulted to be Fleurent and Ferland's[8] genetic-hybrid algorithm. On the other hand, on random, regular, and unstructured problems the performance of our HAS-QAP algorithm resulted less competitive and taboo searches are still the best methods. So far, the most interesting applications of ant colony optimisation were limited to symmetric, asymmetric and constrained traveling salesman problems (Dorigo and Gambardella[14], Gambardella and Dorigo[15]). With this paper we have widened the spectrum of successful applications of ant colony algorithms to structured quadratic assignment problems.

Acknowledgements — Marco Dorigo is a Research Associate with the Belgian FNRS. This research has been funded by the Swiss National Science Foundation contract 21–45653.95 titled "Cooperation and learning for combinatorial optimization."

REFERENCES

1 Koopmans T C and Beckmann M J (1957). Assignment problems and the location of economics activities. *Econometrica* **25**: 53–76.

2 Shani S and Gonzalez T (1976). P-complete approximation problems. *Journal of the ACM* **23**: 555–565.

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

4 Connolly D T (1990). An improved annealing scheme for the QAP. *Eur J Op Res* **46**: 93–100.

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

6 Battiti R and Tecchiolli G (1994). The reactive tabu search. *ORSA J on Computing* **6**: 126-140.

7 Sondergeld L and Voß S (1996). A star-shaped diversification approach in tabu search. In: Osman I H and Kelly J P (eds). *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers: Boston/London/Dordnecht, pp 489–502.

8 Fleurent C and Ferland J (1994). Genetic hybrids for the quadratic assignment problem. *DIMACS Series in Mathematics and Theoretical Computer Science* **16**: 190–206.

9 Cung V-D, Mautor T, Michelon P and Tavares A (1997). A scatter search based approach for the quadratic assignment problem. Proceedings of the *IEEE International Conference on Evolutionary Computation and Evolutionary Programming.* (ICEC' 97), Indianapolis, USA, pp 165-170.

10 Dorigo M, Maniezzo V and Colorni A (1991). The ant system: An autocatalytic optimizing process. Technical Report 91-016, Dipartimento Elettronica e Informazione, Politecnico di Milano, Italy.

11 Dorigo M, Maniezzo V and Colorni A (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics–Part B* **26**: 29–41.

12 Dorigo M (1992). Ottimizzazione, apprendimento automatico, ed algoritmi basati su metafora naturale (Optimisation, learning and natural algorithms). Doctoral dissertation, Dipartimento Elettronica e Informazione, Politecnico di Milano, Italy.

13 Beckers R, Deneubourg J L and Goss S (1992). Trails and U-turns in the selection of the shortest path by the ant Lasius niger. *Journal of Theoretical Biology* **159**: 397–415.

14 Dorigo M and Gambardella L M (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* **1**: 53–66.

15 Gambardella L M and Dorigo M (1997). HAS-SOP: Ant colony optimization for sequential ordering problems. Tech. Rep. IDSIA–11–97, IDSIA, Lugano, Switzerland.

16 Burkard R E and Rendl F (1983). A thermodynamically motivated simulation procedure for combinatorial optimization problems. *Eur J Op Res* **17**: 169–174.

17 Burkard R E, Karisch S and Rendl F (1991). QAPLIB — A quadratic assignment problem library. *Eur J Op Res* **55**: 115–119.
Electronic update: http://www.diku.dk/~karisch/qaplib/ (29.1.1997).

18 Liggett R S (1981). The quadratic assignment problem: An experimental evaluation of solution strategies. *Mgmt Sci* **27**: 442–458.

19 Scriabin M and Vergin R C (1975). Comparison of computer algorithms and visual based methods for plant layout. *Mgmt Sci* **22**: 172–181.

20 Burkard R E and Çela E (1996). Quadratic and three-dimensional assignments: An annotated bibliography. Technical report 63, Discrete Optimisation Group, Technische Universität Graz, Austria.