Adaptive Memories for the Quadratic Assignment Problem

Éric D. Taillard, Luca M. Gambardella
IDSIA, Corso Elvezia 36, CH-6900 Lugano, Switzerland.
{eric, luca}@idsia.ch
IDSIA–87–97 Technical Report

Abstract.

The paper proposes, compares and analyses different memory-based meta-heuristics for the quadratic assignment problem (QAP). Two of these methods (FANT and GDH) are new while two others (HAS-QAP and GTSH) are among the best for structured QAP instances. These methods are based on ant systems and genetic algorithms and they are presented under a unified general scheme, called adaptive memory programming (AMP). However, they use different types of memory and different improving procedures. Two new memoryless methods (VNS-QAP and RVNS-QAP) based on variable neighbourhood search are also proposed and compared with adaptive memory procedures.

Key words: Quadratic assignment, adaptive memory programming, ant system, genetic algorithm.

1. Introduction.

1.1. Adaptive Memory Programming.

The concept of adaptive memory programming (AMP) has been proposed for grouping a number of generic optimization techniques for combinatorial problems. Among these techniques, let us mention genetic algorithms, taboo search and ant systems. Some of these approaches have been proposed for a long time and have been improved and modified a lot since their first proposition. As shown in Taillard et al. (1997) overview paper, the evolution of these techniques has been done in the way of a mutual rapprochement and it becomes difficult to classify a given method in a precise category. Indeed, recent and efficient methods often merge components of many techniques (see, e. g. Fleurent & Ferland (1994), Cung et al. (1997), Gambardella, Taillard & Dorigo (1997), Stützle & Hoos (1997) for applications to the quadratic assignment problem). A number of the best performing heuristic methods are driven by the same common features that are the use of memory and a local search procedure. So, we propose to group and unify all these approaches under the term of AMP, standing for *adaptive memory programming* (or adaptive memory procedure, depending on the context) that is perfectly adapted for such a learning process. Very shortly, an AMP works as follows: a new solution of the problem is constructed by first creating an intermediate solution using informations contained in the memory and then this intermediate solution is improved with a local search procedure. Finally, the improved solution is used to update the memory and the process is iterated. The general scheme of an AMP is given in Figure 1.

```
1) Initialise the memory.
2) Repeat, until a stop criterion is satisfied:
  2a) Construct a new provisory solution, using the information
      contained in the memory.
  2b) Improve the provisory solution with a local search.
  2c) Update the memory.
```

Figure 1 : *General scheme of an adaptive memory programme.*

Starting from this general scheme, it is possible to define various methods, depending on the memory implementation, the constructive procedure, the improving method and the memory updating process. Table 1 provides the main components of the methods discussed in the paper: their name, reference in which they have been proposed, memory implementation, intermediate solution generator and improving procedure.

Name	FANT	HAS-QAP	GDH	GTSH	VNS-QAP	RVNS-QAP
Reference	This paper	Gambardella, Taillard & Dorigo (1997)	This paper	Fleurent & Ferland (1994)	This paper	This paper
Memory	Statistics matrix on solutions	Statistics and population of solutions	Population of solutions	Population of solutions	None	None
Generator	Construc-tive	Local modifi-cations of solu-tions	Cross-over	Cross-over	Random jump	Random jump
Improving procedure	Fast local search	Fast local search	Fast local search	Taboo search	Fast local search	None

Table 1 : *Overview of the basic working principles of the methods studied in this paper.*

This paper present two new AMPs for the quadratic assignment problem (QAP). The first one, referred as FANT in the following, is based on the principle of ant colonies (Dorigo, Maniezzo and Colorni, 1991, Dorigo, 1992) for memory management and intermediate solution generator and is hybridised with a fast local search presented in this paper. The second AMP proposed in this paper, GDH, is a genetic algorithm that uses the cross-over operator of Tate and Smith (1995) for generating intermediate solutions and the same fast local search as FANT as improving procedure.

These new methods are compared with two among the best (AMP) methods developed for the QAP: The genetic-taboo search hybrid algorithm of Fleurent and Ferland (GTSH, 1994) and the hybrid ant system of Gambardella, Taillard and Dorigo (HAS-QAP, 1997), also based on a combination of ant systems and a fast local search. The new AMPs proposed in this paper are shown to be among the best methods for solving structured instances of QAP that derives form practical applications, where the data matrices describing the problem have very variable entries. For instances where the matrix entries are more uniform, there exist taboo searches, as those of Battiti and Tecchiolli (1994) or Taillard (1991) that are more competitive, as clearly shown in Taillard (1994) and Gambardella, Taillard and Dorigo (1997). So, we concentrate our analysis on structured problems for which a learning phenomenon can effectively be observed.

In order to understand the importance of the memory component, we also propose two memoryless methods (VNS-QAP and RVNS-QAP) that are based on variable neighbourhood search (Hansen and Mladenovic, 1997). We consider a standard VNS that incorporates the fast improving method used in the AMPs, and RVNS, a reduced variant that does not apply this improving method. RVNS is able to perform a number of iterations $O(n^2)$ larger than VNS in the same amount of time (for a problem of size n). This is the first time that VNS is applied to the QAP.

1.2 The quadratic assignment problem (QAP).

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

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

We denote by $\Pi(n)$ the set of permutations of n elements. Shani and Gonzalez (1976) 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 optimization problems can be exactly solved for relatively large instances, as exemplified by the travelling salesman problem, 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, 1994). In the present paper, we report new best solution values for a number of these large instances.

The paper is structured as follows: Section 2 presents the fundamentals of ant systems and our new FANT method for the QAP. This section will also review the HAS-QAP method of Gambardella, Taillard and Dorigo (1997). Section 3 presents the fundamentals of genetic algorithms and our new GDH method, as well as the GTSH method of Fleurent and Ferland (1994). These four methods will be presented under the AMP perspective: According to the general frame of AMP shown in Figure 1, an AMP implementation can be specified by four components:

a) The memory structure.

b) A procedure for building a provisory solution.

c) A procedure for improving the provisory solution.

d) The way the memory is updated.

We are going to describe the AMP methods by specifying the choices made for implementing these four components. Section 4 presents the fundamentals of variable neighbourhood search and our new VNS-QAP and RVNS-QAP methods. Finally, Section 5 analyses and discusses the numerical performances of all these methods.

2. Ant systems.

The origin of ant systems is to imitate the behaviour of ants searching for food. This meta-heuristic has been proposed by Dorigo, Maniezzo and Colorni (1991) and Dorigo (1992). Ants are finding sources of food in the following way: First, they explore the area surrounding their nest in a random manner. While they are moving, the ants left a pheromone (chemical trace) on the floor, in such a way that they can find their way back to the nest. When they find a source of food, the ants bring food back to the nest following the pheromone traces, leaving additional pheromone during the return trip. After a while, the paths between the nest and sources of food will be indicated by an amount of pheromone in relation with the length of the path. Indeed, short paths will be travelled at an higher rate than long ones and the amount of pheromone will grow faster on the short ways. Therefore, the ants are able to optimize their paths by this process.

A similar process can be transposed to combinatorial optimization: solutions of the problem are built using a statistics on solutions previously generated. This statistics play the rôle of the pheromone traces and it gives an higher weight to the best solutions. After a while, it is observed that such a procedure is able to build solutions of better quality than a procedure guided by partial objective function evaluations only.

2.1 FANT: Fast ant system for the QAP.

Now, let us see how an ant system can be implemented for the QAP in practice. Our FANT method presents however a number of differences regarding to basic ant systems as described by Dorigo, Maniezzo and Colorni (1991). First, we have added a local search for improving the solution built by the ant. This has been identified as very promising for other problems such as the travelling salesman problem (Dorigo and Gambardella, 1997) or the sequential ordering problem (Gambardella and Dorigo, 1997).

Another difference is the statistics that is computed. Indeed, this statistics gives more and more weight to the best solution found so far as the search progresses. This makes the search to converge faster to good solutions. However, this may also causes the search to be trapped in local optima. In order to avoid this situation, a *diversification* mechanism that reset the memory has been developed. Finally, FANT does not use a population of artificial agents.

Let us now present the FANT method under the AMP perspective, i. e. by specifying its four components: The memory structure, the constructing procedure, the improving procedure and the way the memory is updated.

a) Memory implementation.

The basic idea of our method is to evaluate a priori the interest of setting $\pi_i = j$ in a solution (= permutation) π. Therefore, our memory is principally constituted by a matrix T of size $n \times n$ whose entry τ_{ij} measures the preference of setting $\pi_i = j$.

From an ant system point of view, this matrix represents the pheromone trail left by the ants. The entries of this matrix consist in a statistics derived from the solution previously generated by the search process.

b) Generation of a provisory solution.

For generating a provisory solution μ, we use a constructive method that chooses the elements of μ successively, in a random order and with a probability proportional to the values contained in the T matrix. More formally, the constructive method is presented in Figure 2:

1) $I = \varnothing$, $J = \varnothing$
2) While $|I| < n$ repeat:
   2a) Choose i, randomly, uniformly, $1 \le i \le n$, $i \notin I$.
   2b) Choose j, randomly, $1 \le j \le n$, $j \notin J$, with probability $\tau_{ij} / \sum_{1 \le k \le n, k \notin J} \tau_{ik}$ and set $\mu_i = j$.
   2c) $I = I \cup \{i\}$, $J = J \cup \{j\}$

Figure 2 : *Construction of a provisory solution.*

c) Improvement procedure.

The provisory solution μ generated at the previous step is not so good, generally; at the first iteration, μ is just a random permutation. So, we apply to μ the first steps of a first improving neighbour procedure. More formally, we repeat the procedure of Figure 3 twice, where $\Delta(\pi, i, j)$ is the difference in the objective function value when exchanging the elements π_i and π_j in π.

1) $I = \varnothing$.
2) While $|I| < n$ repeat:
   2a) Choose i, randomly, uniformly, $1 \le i \le n$, $i \notin I$.
   2b) $J = \{i\}$
   2c) While $|J| < n$ repeat:
      2c1) Choose j, randomly, $1 \le j \le n$, $j \notin J$.
      2c2) If $\Delta(\pi, i, j) < 0$, exchange π_i and π_j in π.
      2c3) $J = J \cup \{j\}$
   2d) $I = I \cup \{i\}$.

Figure 3 : *Fast descent procedure (to be repeated twice).*

The evaluation of $\Delta(\pi, i, j)$ can be performed in $O(n)$ using the formula:

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

Therefore this procedure can be executed in $O(n^3)$; it does not necessarily returns a local optimum, but it is fast and may produce different solutions when starting with the same initial, not locally optimal solution. From now on, we are going to call π the improved solution.

d) Memory updates.

The statistics the memory stores uses two parameters, r and r^*, that represent the re-enforcement of the matrix entries corresponding to the solution produced at step 2c of the AMP and, respectively, π^*, the best solution produced by the algorithm. During the entire process, we set $r^* = 4$ while r may vary. At the beginning $r = 1$ and we set $\tau_{ij} = r$, $1 \le i, j \le n$, meaning that the memory does not contain any information initially. Usually, the entries of matrix T are updated as follows:

1) For i = 1 to n do:

  1a) $\tau_{i\pi_i} = \tau_{i\pi_i} + r$

  1b) $\tau_{i\pi_i^*} = \tau_{i\pi_i^*} + r^*$

Where π is the solution produced at the current iteration and π^* the best solution produced so far. In two cases, this update is done in another way:

1) If π^* has been improved, then r is reset to 1 and all the entries of T are set to 1. The aim of this resetting is to intensify the search around π^* by giving less importance to the past of the search.

2) If the provisory solution μ generated at step 2b is equal to π^*, then r is increased by one unit and all the entries of T are set to r. This situation occurs when π^* has not been improved for a large number of iterations, meaning that the re-enforcement of the entries corresponding to π^* is too high. The aim of this is to diversify the search when the information contained in T is not very different from π^*.

Now, let us review a completely different way of implementing an ant system for the QAP that has been proposed by Gambardella, Taillard and Dorigo (1997).

2. 2. HAS-QAP: Hybrid ant system for the QAP.

The method of Gambardella, Taillard and Dorigo (1997), called HAS-QAP for short, is described as an hybrid ant system-local search. Indeed, traditional ant systems build a solution with the memory from scratch. In HAS-QAP, the memory is used to modify existing solutions, in the spirit of local search. Moreover, HAS-QAP uses the fast descent procedure of Figure 3.

a) Memory implementation.

There are two different memories in HAS-QAP. The first one is a matrix that records a statistics on the best solutions produced by the search. The second one is a population of m (= 10) solutions, in addition to the best solution produced by the search. Initially, all the entries of the matrix are identical and the solutions memorised are random permutations improved with the fast local search procedure of Figure 3. From a memory point-of-view, this method is between the genetic methods presented further that use a population of solutions only and FANT that uses a statistics-matrix only.

b) Provisory solution.

Instead of building a completely new solution, HAS modify all the solutions in memory by swapping a number of times two elements of the solutions. The first element is chosen randomly and uniformly and the second one in a probabilistic way, considering the values contained in the statistics-matrix.

c) Improvement procedure.

The provisory solutions are improved with the fast descent procedure of Figure 3.

d) Memory update.

Normally, the entries τ_{ij} of the statistics-matrix $T = (\tau_{ij})$ are updated as follows: First, $\tau_{ij} = (1 - \alpha)\, \tau_{ij}$ $(1 \le i, j \le n)$ where α is a parameter. The effect of this setting is to lower the value of all the entries of the matrix, simulating the evaporation of pheromone trails of real ants. Then, few entries of the matrix are re-enforced by setting: $\tau_{i\pi_i^*} = \tau_{i\pi_i^*} + \alpha / f(\pi^*)$ $(1 \le i \le n)$, where π^* is the best solution found by the search so far.

The m solutions of the population are replaced by the improved solutions. Exceptionally, the memory is updated in another way, in two conditions:

1) If π^*, the best solution has not been improved during a number of iterations, the memory is completely re-initialised: All the entries of the matrix are set to the same value and $m - 1$ solutions of the population are randomly generated while the m^{th} becomes π^*. The aim of this special treatment is to diversify the search.

2) If π^*, the best solution has just been improved, then the solution obtained after applying the fast descent procedure replaces those in memory only if it is better. Otherwise, the solution initially in memory is not modified. The aim of this special treatment is to intensify the search around good solutions.

3. Genetic algorithms.

Genetic algorithms also simulate a natural process: those of the evolution of species that sexually reproduce and has been proposed by Holland (1975). During sexual reproduction, a new individual, defined by its genetic patrimony, is created by combining half of the genetic patrimony of its parents. Eventually, the genetic patrimony is altered by a mutation. If the new individual inherits from good characteristics, its survival probability is higher, as well as the number of times it can reproduce itself. So, good characteristics are more disseminated and the population tends to contain better and better individuals.

This process can be used to optimize combinatorial problems as well: solutions play the rôle of individuals, sexual reproduction is simulated by a cross-over operator that mixes two solutions to create a new one that is eventually altered by a mutation operator. Let us see shortly how this can be implemented for the QAP by first reviewing the GTSH method of Fleurent and Ferland (1994). Our new GDH method can be viewed as a variant of GTSH. As above, the methods are presented under the AMP perspective.

3. 1. Genetic-Taboo Search Hybrid (GTSH).

The GTSH method has been proposed by Fleurent and Ferland (1994). It can be presented as a genetic algorithm, using a special cross-over operator specially designed for permutations (due to Tate and Smith, 1995) and a mutation operator that consists in performing $4n$ steps of a taboo search (due to Taillard, 1991). In the AMP terminology, GTSH can be presented as follows:

a) Memory implementation.

The memory is constituted by 100 solutions (permutations). Initially, these solutions are obtained by 100 taboo search runs performing $4n$ iterations each and starting with random initial solutions. For problem of small size ($n < 50$), Taillard (1994) uses a population of $2n$ solutions instead of 100. This speeds-up the search notably. In the following, we consider this improved version of the algorithm of Fleurent and Ferland with a population of $min(100, 2n)$ solutions.

b) Generation of a provisory solution.

The provisory solution is build as follows. First, two permutations are selected from the population. The probability of selecting the k^{th} worst solution of the population is $2k/s(s-1)$, where s is the size of the population. The cross-over operator of Tate and Smith (1995) is applied to the selected solutions to produce the provisory solution. This operator proceeds in three phases: 1) The elements common in both selected permutations are copied in the new solution at the same place. 2) If not already chosen, the other elements are randomly chosen from the selected permutations, at the same position. 3) The unfilled positions are randomly chosen in order to complete the solution.

c) Improvement procedure.

The improvement procedure is the taboo search of Taillard (1991). It is run for $4n$ iterations, starting from the provisory solution.

d) Memory update.

The new solution is inserted in the memory and the worst solution is removed from the memory.

3. 2. Genetic-Descent Hybrid method (GDH).

To better study the influence of the knowledge brought by the memory and by taboo search, we have slightly modified the algorithm of Fleurent and Ferland and replaced the taboo search by our fast descent method presented in Figure 3. So, we can better compare the matrix-memory learning process of FANT with a population-memory learning process since the improving procedures embedded in FANT and GDH are the same. Moreover, the comparison of both methods is more reliable since most of the computing time is spent in the improvement procedure; thus, the computation effort can be measured in terms of number of calls to the improvement procedure rather than in seconds.

4. Variable neighbourhood search (VNS).

VNS is a recent meta-heuristic proposed by Hansen and Mladenovic (1997) that presents a number of similarities with the strategic oscillation, the taboo alternating paths and taboo thresholding of Glover (1986, 1990, 1995). The basic idea of VNS is to use several neighbourhood structures instead of a unique one, as it is often the case in many local search implementations. For the QAP, we define the neighbourhood $N_k(\pi)$ of a solution π as all the permutations that can be obtained by applying k swaps to π. Our basic VNS for the QAP can be described as follows:

```
1) Choose π* randomly, set k = 1.
2) Repeat for I iterations:
  2a) Choose μ, randomly in N_k(π*)
  2b) Apply the fast descent method to μ to obtain π
  2c) If f(π) < f(π*) then set π* = π and k = 1
      Else set k = (k modulo K_max) + 1
```

Figure 4 : *Variable neighbourhood search for the QAP.*

The RVNS-QAP method is based on the same scheme but the descent procedure is omitted in step 2b. In our implementation this allows to perform roughly $n^2/2$ iterations in the same time than VNS-QAP requires for performing one iteration. For VNS-QAP, we have chosen $K_{max} = n$ while for RVNS-QAP, we have chosen $K_{max} = 5$ since we have observed that an improving solution in N_k was never obtained if k was larger than 5.

5. Numerical results.

5.1. Analysis of the methods.

The four AMPs presented above, FANT, HAS-QAP, GDH and GTSH, are very different, even if they are based on the same basic concepts. Considering in addition the no-memory methods VNS-QAP and RVNS-QAP, we can try to answer few questions in this section:

— What is the benefit of using a memory.

— What type of memory learns better or faster.

— Which procedure producing provisory solutions is best.

— What knowledge is brought by the improving procedure.

It is clear that the answers to these questions are not independent each others since the choice of the memory type implies a limitation in the choice of the provisory solution generation. Four of these methods, FANT, GDH, VNS and HAS-QAP use the same improving procedure. The complexity of the last, as well as those of the taboo search used in GTSH is $O(n^3)$. The other components of these methods (provisory solution generation, update of the memories) have a lower complexity level. So, it is natural to express the computation effort in number of calls to the improvement procedure, the time spent outside this procedure being negligible.

In our implementations, we have observed that $4n$ (respectively n^2) iterations of the taboo search of Taillard (1991) (respectively RVNS-QAP) take almost 8 (respectively 2) times more time than our fast improving procedure. Therefore, in tables 2 to 4, we allow less iterations to the GTSH method and more to the RVNS-QAP, in order to keep a parity in computation time.

To compare our methods, we consider a set of about 30 well known QAP instances from the QAPLIB compiled by Burkard, Karish and Rendl (1991). The name of these instances contains a number corresponding to their size. Most of these problems have data matrices with very variable entries and are issued from practical applications or have been randomly generated with non uniform laws, imitating the distributions observed in real world problems. However, the *nug..* and *sko..* instances are more regular. The selected instances are particularly hard to solve by local searches, as shown experimentally by Taillard (1994) and Gambardella, Taillard and Dorigo (1997) or by theoretic complexity measures discussed in Angel and Zissimopoulos (1997). In Table 2 (respectively Tables 3 and 4), we compare FANT, HAS-QAP, GDH, GTSH, VNS-QAP and RVNS-QAP when they run for the same computing time, corresponding to 10 (respectively 100 and 1000) calls to the fast descent procedure. Comparisons are made by averaging the quality of the solutions produced by these methods (measured in per cent above the best solution known) over 10 independent runs of each method. The exact computing time is given for FANT only, (seconds on Sun Sparc 5); the times for the other methods are almost the same.

For very short runs (Table 2), only FANT has had time to initialise the memory and to begin the learning process. The other AMPs are still in the initialisation phase: HAS-QAP and GHD have just

Problem name	Best solution value known	FANT	HAS-QAP ≡ GDH	GTSH ≡ TT	VNS-QAP	RVNS-QAP	Time FANT [s]
bur26a	5426670	**0.129**	**0.158**	0.250	0.194	0.243	0.47
bur26b	3817852	0.230	0.184	0.540	0.210	0.423	0.45
bur26c	5426795	**0.052**	0.066	0.341	0.124	0.381	0.46
bur26d	3821225	**0.046**	0.064	0.587	0.057	0.361	0.46
bur26e	5386879	**0.040**	0.042	0.123	0.107	0.216	0.46
bur26f	3782044	0.149	**0.025**	0.404	0.145	0.540	0.45
bur26g	10117172	0.053	**0.030**	0.384	0.210	0.211	0.46
bur26h	7098658	0.017	**0.014**	0.311	0.380	0.445	0.46
els19	17212548	**3.122**	6.426	32.062	14.204	19.180	0.21
kra30a	88900	**4.758**	5.447	5.404	5.250	6.578	0.74
kra30b	91420	3.373	**3.345**	2.330	3.538	4.747	0.83
nug20	2570	2.319	2.537	**1.424**	2.093	3.829	0.23
nug30	6124	2.283	2.469	2.286	**1.646**	3.266	0.78
sko42	15812	2.127	2.657	**1.551**	2.222	3.305	2.38
sko49	23386	1.938	2.091	1.549	**1.492**	2.797	4.01
sko56	34458	2.003	1.918	**1.621**	1.688	2.294	6.19
sko64	48498	1.801	2.052	**1.005**	1.719	2.441	9.52
sko72	66256	1.781	1.975	**1.481**	1.778	2.286	13.82
sko81	90998	1.779	1.732	1.491	**1.420**	1.753	20.18
sko90	115534	1.627	1.604	**1.346**	1.843	1.824	28.10
tai20b	122455319	**1.690**	2.887	19.611	7.185	11.495	0.25
tai25b	344355646	**2.584**	3.040	15.931	6.008	11.150	0.48
tai30b	637117113	**1.977**	3.566	15.716	4.523	8.282	0.86
tai35b	283315445	2.855	**1.989**	6.491	5.430	7.631	1.43
tai40b	637250948	**4.226**	4.702	8.740	7.453	8.979	2.32
tai50b	458821517	2.801	**2.302**	6.247	5.073	4.403	4.65
tai60b	608215054	2.918	**2.444**	7.818	3.799	7.439	8.31
tai80b	818415043	**3.704**	4.067	4.796	3.789	5.351	20.42
Average		**1.871**	2.137	5.007	2.985	4.352	

Table 2 : *Performances of the methods on short runs (time equivalent to 10 calls to the improving procedure)*

produced the 10 first solutions of their population, so both methods are equivalent; GTSH has just had the time to produce the first solution of its population and is therefore equivalent to running the taboo search of Taillard (1991) for $4n$ iterations (TT). For these very short runs we see that our new FANT often produces the best results (identified by bold characters in the tables). This can be explained by the fact that its learning process is the fastest. However, for the problems of the type *nug*.. and *sko*.., TT is better while it can produce very bad results on irregular and more structured instances, like *els*.. and *tai..b.*

Although RVNS-QAP is able to perform a much larger number of iterations per unit of time than VNS-QAP, it is almost never better than the last. On the average, our implementation of VNSs are better than TT for short runs and irregular problems.

Problem name	FANT	HAS-QAP	GDH	GTSH	VNS-QAP	RVNS-QAP	Time FANT [s]
bur26a	0.0449	**0.0275**	0.0581	0.0903	0.055	0.198	4.6
bur26b	0.0717	0.1065	0.0571	0.1401	**0.053**	0.346	4.5
bur26c	**0.0000**	0.0090	0.0021	0.0112	0.001	0.347	4.6
bur26d	0.0022	**0.0017**	0.0039	0.0059	0.003	0.223	4.5
bur26e	0.0068	**0.0039**	0.0059	0.0066	0.005	0.121	4.6
bur26f	0.0007	**0.0000**	0.0034	0.0097	0.001	0.357	4.5
bur26g	0.0025	**0.0002**	0.0027	0.0094	0.003	0.153	4.6
bur26h	**0.0003**	0.0007	0.0021	0.0063	0.001	0.305	4.5
els19	0.5148	0.9225	1.6507	2.2383	**0.421**	14.056	2.0
kra30a	2.4623	1.6637	2.9269	2.0070	1.660	5.139	7.4
kra30b	0.6607	**0.5043**	1.1748	0.6082	0.677	4.137	8.3
nug20	0.6226	**0.1556**	0.4825	0.1774	0.638	2.903	2.2
nug30	0.6172	0.5650	1.1920	**0.3971**	0.686	2.972	7.8
sko42	0.8158	**0.6539**	1.2168	0.6964	0.971	3.045	23.8
sko49	0.7740	0.6611	1.4530	**0.4936**	0.547	2.518	40.0
sko56	1.0941	0.7290	1.4272	**0.5869**	0.923	2.215	61.9
sko64	0.9213	**0.5035**	1.2739	0.6678	0.804	2.310	95.2
sko72	0.8826	**0.7015**	1.4224	0.7130	0.762	2.166	138.2
sko81	0.9950	**0.4925**	1.3218	0.7125	0.561	1.666	201.8
sko90	0.9241	**0.5912**	1.2518	0.7606	0.842	1.773	280.9
tai20b	0.1810	0.2426	**0.1807**	1.0859	0.226	1.252	2.4
tai25b	**0.1470**	0.1326	0.3454	1.9563	0.196	7.792	4.8
tai30b	**0.2550**	0.2603	0.3762	3.2438	0.510	4.197	8.6
tai35b	0.3286	0.3429	0.7067	1.5810	**0.248**	5.795	14.3
tai40b	1.3401	**0.2795**	0.9439	3.9029	1.559	7.421	23.0
tai50b	0.5412	**0.2906**	1.1281	1.6545	0.642	4.063	46. 5
tai60b	0.6699	**0.3133**	1.2756	2.6585	0.880	6.309	83.0
tai80b	1.4379	**1.1078**	2.3015	2.7702	1.569	4.909	204.1
Average	0.583	**0.402**	0.864	0.936	0.552	3.168	

Table 3 : *Comparison on medium runs (100 calls to the improving procedure)*

For medium runs (Table 3), HAS-QAP is the best method. But FANT and VNS-QAP remains very competitive while GTSH is still in the initialisation phase. For instances of size 50 or more, GDH has just finished its initialisation and the learning process has not begun. So, the performances of GDH are not as good as FANT and HAS-QAP but better than GTSH. However, for the *sko..* instances GTSH is better than GHD, meaning that the use of taboo search seems to be an advantage.

RVNS-QAP performs poorly on medium and long runs: the solution quality is almost the same as the quality for short runs. This means that the use of a descent method helps a lot in finding good solutions, if enough computing time is available.

For long runs (Table 4), FANT and VNS-QAP are generally surpassed by HAS-QAP that had time to develop its learning process. It seems that the addition of a small population of solutions is

Problem name	FANT	HAS-QAP	GDH	GTSH	VNS-QAP	Time FANT [s]
bur26a	0.0253	**0**	0.0345	0.0179	0.001	46
bur26b	0.0171	**0**	0.0358	0.0304	**0**	45
bur26c	**0**	**0**	**0**	**0**	**0**	(2.9)
bur26d	**0**	**0**	0.0004	0.0002	**0**	(9.2)
bur26e	**0**	**0**	0.0012	0.0005	**0**	(6.6)
bur26f	**0**	**0**	**0**	**0**	**0**	(4.2)
bur26g	**0**	**0**	0.0010	**0**	**0**	(8.2)
bur26h	**0**	**0**	0.0010	0.0012	**0**	(2.0)
els19	**0**	**0**	1.4425	**0**	**0**	(1.1)
kra30a	1.0304	0.6299	1.0787	**0.3597**	0.802	74
kra30b	0.0919	0.0711	0.1652	**0.0641**	0.153	83
nug20	0.1401	**0**	**0.0156**	**0**	**0**	22
nug30	0.2515	0.0980	0.1600	**0.0431**	0.271	78
sko42	0.2429	**0.0759**	0.0873	0.1897	0.186	237
sko49	0.2395	**0.1411**	0.1890	0.2037	0.220	400
sko56	0.3216	**0.1010**	0.1544	0.3023	0.600	619
sko64	0.1942	**0.1291**	**0.1423**	0.3820	0.421	951
sko72	0.3450	0.2765	**0.1980**	0.3876	0.409	1382
sko81	0.2637	**0.1437**	0.1666	0.4342	0.337	2018
sko90	0.3966	0.2311	**0.1949**	0.4342	0.452	2809
tai20b	0.0905	0.0905	0.0905	**0**	0.045	24
tai25b	**0**	**0**	0.0139	0.0605	0.007	(13)
tai30b	0.0003	**0**	0.0290	0.1426	0.001	85
tai35b	0.0373	0.0256	**0.0187**	0.2639	0.072	143
tai40b	0.2016	**0**	0.2038	0.7383	0.001	228
tai50b	0.2063	0.1916	**0.0086**	0.6433	0.183	466
tai60b	0.2478	0.0483	**0.0089**	0.8313	0.076	830
tai80b	0.8245	0.6670	**0.2246**	1.7358	0.907	2041
Average	0.185	**0.104**	0.167	0.216	0.184	

Table 4 : *Comparison long runs (1000 calls to the improving procedure)*

useful at long term. The pertinence of this remark is strengthened by the fact that GDH seems to be the best method for *tai..b* instances. For *sko..* instances, HAS-QAP and GDH are clearly the best methods among those compared in Table 3.

In Table 4, we put in parentheses the average computing time of FANT to obtain the best solution known, when the 10 runs succeeded in finding this solution. The time to complete the 1000 iterations can be approximated by 10 times the time to complete 100 iterations (Table 3).

It seems that the learning process obtained with a population of solutions is very efficient at long term and the use of a simple descent procedure embedded in the process is more efficient than a basic taboo search. So, a natural question that arises, is to know whether the use of an improving procedure is necessary or not, since this procedure is time consuming, while the time spent in this procedure could be used to allow more iterations. The answer is clearly no. RVNS-QAP is worse than VNS-QAP and numerical experiments (not reported here) obtained by removing the improving procedure of the GTSH or GDH methods have shown that the population converges toward a solution that is very bad even by considering a much larger population: with a population of 500 solutions, the solutions produced by the method without improving procedure are worse than those of Table 2 while using a computation time larger than those of Table 4. So, we can say that each component of AMP is necessary. The improving procedure is not able to find good solutions if it is not guided by a learning process that produces solutions with good characteristics and the memory does not succeed in learning if it does not dispose of a myopic improving procedure.

5. 2. New best solutions known.

Our new methods FANT and GDH are very aggressive and fast. So, they are adapted for dealing with large size instances where a large number of iterations are required. We have applied a

m	Best known	*m*	Best known	*m*	Best known	*m*	Best known
3	7810	35	4890132	67	21461130	99	52660116
4	15620	36	5222296	68	22271676	100	53838088
5	38072	37	5565236	69	23086332	101	55014262
6	63508	38	5909202	70	23898390	102	56208480
7	97178	39	6262248	71	24742496	103	57417112
8	131240	40	6613472	72	25570996	104	58629516
9	183744	41	7002794	73	26423856	105	59854744
10	242266	42	7390586	74	27276468	106	61084902
11	304722	43	7795062	75	28154258	107	62324634
12	368952	44	8219428	76	29001308	108	63582416
13	457504	45	8677670	77	29928042	109	64851966
14	547522	46	9134172	78	30820660	110	66120830
15	644036	47	9575736	79	31736576	111	67392724
16	742480	48	10016256	80	32658876	112	68666416
17	878888	49	10523016	81	33571824	113	69984758
18	1012990	50	11017342	82	34497444	114	71304194
19	1157992	51	11517060	83	35447648	115	72630764
20	1305744	52	12019082	84	36409340	116	73962220
21	1466210	53	12558226	85	37379574	117	75307424
22	1637794	54	13098850	86	38393344	118	76657014
23	1820052	55	13668486	87	39403048	119	78015914
24	2010846	56	14238840	88	40472972	120	79375832
25	2216160	57	14798834	89	41567886	121	80756852
26	2426606	58	15395948	90	42621464	122	82138768
27	2645436	59	15981086	91	43704936	123	83528554
28	2871704	60	16575644	92	44759294	124	84920540
29	3122510	61	17202312	93	45891688	125	86327812
30	3373854	62	17824828	94	46988852	126	87736646
31	3646344	63	18435790	95	48108960	127	89150166
32	3899744	64	19050432	96	49182368	128	90565248
33	4230950	65	19859448	97	50344050		
34	4560162	66	20655396	98	51486642		

Table 5 : *Best solution values known to problems instances grey_16_16_m.*

GDH with a larger population of solutions to instances tai150b and, respectively tho150; the process has converged to the new best solution value 49889664 and, respectively 8133642. after about 100000 iterations.

In Table 5, we give the values of the best solutions we have found with very long runs (up to 100000 iterations) of our FANT and GDH methods for QAP instances corresponding to the elaboration of grey frames. This type of instances is described in Taillard (1994) under the name grey_n_1_ n_2_m, where m is the density of the grey one wants to produce ($0 \le m \le n = n_1 \cdot n_2$). So, for a frame of size $n_1 \times n_2$, there are $n_1 \cdot n_2$ instances of size $n = n_1 \cdot n_2$ to solve.

In fact, solutions for densities $m > n_1 \cdot n_2/2$ can be obtained by symmetry from densities $m < n_1 \cdot n_2/2$. Moreover, for few values of m, the problem is easy; for example for $m = 0, 1, n_1 \cdot n_2$ or $n_1 \cdot n_2 - 1$, any solution is optimal. The problem instance *tai256c* from QAPLIB corresponds to instance grey_16_16_92. To obtain the other instances associated to other values of m, one has to replace the data matrix of the form: $\begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$, where **1** is a sub-matrix of size 92×92 composed of 1 only, by a sub-matrix of size $m \times m$ composed of 1 only. The very special character of this matrix allows to simplify and to shorten a lot the computation, as shown in Taillard (1994).

6. Conclusions.

In this paper, we have proposed FANT, GDH, VNS-QAP and RVNS-QAP new heuristic methods for the QAP. The first methods are based on adaptive memory programming (AMP) while the last ones do not use memory. We group under the generic name of AMP a number of meta-heuristic that are working on the same general principle: 1) A memory stores solutions or characteristics of solutions produced by the search; 2) A procedure generates new provisory solution, consulting the memory; 3) An improving procedure is applied to the provisory solution before updating the memory.

Our new methods are easy to implement and are very efficient for some type of instances. They allow to improve a number of the best solutions known to the largest instances of the literature. These new methods are compared to other methods that are also based on AMP, but using other types of memory and other improving procedures. So, we were able to extract the influence of the various components of AMP: To obtain the best solutions in the shortest time, it seems that FANT is very well adapted; a simple statistics on the solutions produced by the search is convenient. If solutions of higher quality are desired, it seems that keeping a number of solutions in memory can be recommended.

The improving procedure is mandatory if one wants to find good solutions and it must be as fast as possible. The use of a high quality procedure, like a basic taboo search that requires a larger amount of computation time, seems to be counter-productive if the time factor is important.

We have to mention that the methods compared in this paper, FANT, HAS-QAP, GDH, GTSH and VNS-QAP are among the most competitive methods for irregular QAP instances. For other type of instances, none of these method is competitive regarding to other taboo searches. This can be explained by the fact that the learning processes are not adapted for these instances. We can imagine, for example, that the attraction basin of the global optimum is small and hidden by numerous local optima of high quality.

Acknowledgements

This research was supported by the Swiss National Science Foundation project number 21-45653.95.

6. Bibliography

E. Angel and V. Zissimopoulos, "On the landscape ruggedness of the quadratic assignment problem", *International workshop on combinatorics and computer science LIX–CNRS*, Palaiseau, France, September 1997.

R. Battiti and G. Tecchiolli, "The reactive tabu search", *ORSA Journal on Computing 6*, 1994, 126–140.

R. E. Burkard, S. Karisch and F. Rendl, "QAPLIB — A quadratic assignment problem library", *European Journal of Operational Research 55*, 1991, 115–119, electronic update: http://fmt-bhpl.tu–graz.ac.at/~karisch/qaplib, (29. 1. 1997).

A. Colorni, M. Dorigo and V. Maniezzo, "Distributed Optimization by Ant Colonies", *Proceedings of the European Conference on Artificial Life*, Elsevier Publishing, 1991, 134–142.

V.-D. Cung, T. Mautor, P. Michelon and A. Tavares, "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, 1997, 165–170.

D. T. Connolly, "An improved annealing scheme for the QAP", *European Journal of Operational Research 46*, 1990, 93–100.

M. Dorigo, "Ottimizzazione, aprendimento automatico, et algoritmi basati su metafora naturale", Ph. D. dissertation, Dipartimento Elettronica e Informazione, Politecnico di Milano, Italy, 1992.

M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative Learning Approach to the Traveling Salesman Problem", *IEEE Trans. Evolutionary Computing 1*, 1997.

M. Dorigo, V. Maniezzo and A. Colorni, "The Ant System: An Autocatalytic Optimizing Process", *Technical Report 91–016*, Dipartimento Elettronica e Informazione, Politecnico di Milano, Italy, 1991.

M. Dorigo, V. Maniezzo and A. Colorni, “The Ant System: Optimization by a Colony of Cooperating Agents”, *IEE Transactions on System, Man, and Cybernetics –Part B 26*, 1996, 29–41.

C. Fleurent and J. Ferland, “Genetic hybrids for the quadratic assignment problem”, *DIMACS Series in Math. Theoretical Computer Sci. 16*, 1994, 190–206.

L. M. Gambardella and M. Dorigo, “HAS-SOP: Ant colony optimization for sequential ordering problems”, technical report *IDSIA–11–97*, IDSIA, Lugano, Switzerland, 1997.

L. M. Gambardella, E. D. Taillard and M. Dorigo, “Ant colonies for the quadratic assignment problem”, technical report *IDSIA–4–97*, IDSIA, Lugano, Switzerland, 1997.

J. H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, 1975.

F. Glover, “Future Paths for Integer Programming and Links to Artificial Intelligence”, *Computers and Operations Research 13*, 1986, 156–166.

F. Glover, “Simple Tabu Thresholding”, Proceedings of the *Journées du L. I. P. N.*, Paris, 1990, 135–143.

F. Glover, “Tabu Thresholding: Improved Search by Nonmonotonic Trajectories”, *ORSA Journal on Computing 7*, 1995.

P. Hansen and N. Mladenovic, “An introduction to variable neighborhood search”, Presented at the *2nd Metaheuristics International Conference*, Sophia-Antipolis, France, 1997.

T. C. Koopmans and M. J. Beckmann, “Assignment problems and the location of economics activities”, *Econometrica 25*, 1957, 53–76.

S. Shani and T. Gonzalez, “P-complete approximation problems”, *Journal of the ACM 23*, 1976, 555–565.

L. Sondergeld and S. Voß, “A Star-Shaped Diversification Approach in Tabu Search”, in: *Meta-Heuristics: Theory and Applications*, I. H. Osman and J. P. Kelly (editors), Kluwer Academic Publishers, 1996, 489–502.

T. Stützle and H. H. Hoos, “The MAX-MIN Ant System and Local Search for Combinatorial Optimization Problems: Towards Adaptive Tools for Global Optimization”, *Proceedings of the 2nd Metaheuristics International Conference*, Sophia-Antipolis, France, 1997, 191–193.

É. D. Taillard, “Robust taboo search for the quadratic assignment problem”, *Parallel Comput. 17*, 1991, 443–455.

É. D. Taillard, “Comparison of iterative searches for the quadratic assignment problem”, *Location Science 3*, 1995, 87–105.

É. D. Taillard, L. M. Gambardella, M. Gendreau and J.-Y. Potvin “Programmation à mémoire adaptative”, technical report *IDSIA–79–97*, IDSIA, Lugano, Switzerland, 1997.

D. E. Tate and A. E. Smith, “A genetic approach to the quadratic assignment problem”, *Computers and Operations Research 1*, 1995, 855–865.