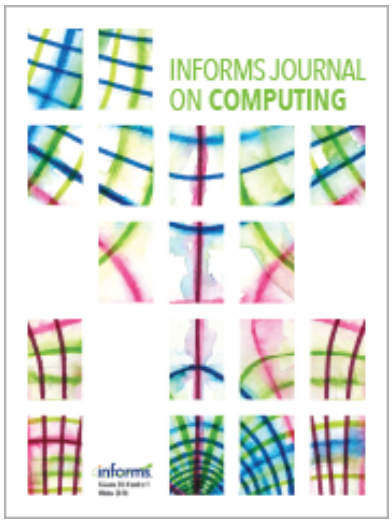
This article was downloaded by: [159.149.103.9] On: 08 December 2019, At: 07:44
Publisher: Institute for Operations Research and the Management Sciences (INFORMS)
INFORMS is located in Maryland, USA



INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem

Vittorio Maniezzo,

To cite this article:
Vittorio Maniezzo, (1999) Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem. INFORMS Journal on Computing 11(4):358-369. https://doi.org/10.1287/ijoc.11.4.358

Full terms and conditions of use: https://pubsonline.informs.org/Publications/Librarians-Portal/PubsOnLine-Terms-and-Conditions

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

© 1999 INFORMS

Please scroll down for article—it is on subsequent pages

With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes.
For more information on INFORMS, its publications, membership, or meetings visit http://www.informs.org

INFORMS Journal on Computing
Vol. 11, No. 4, Fall 1999

0899-1499/99/1104-0358 $05.00
© 1999 INFORMS

Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem

VITTORIO MANIEZZO / *Scienze dell'Informazione, Università di Bologna, Via Sacchi, 3, 47023, Cesena, Italy; Email: maniezzo@csr.unibo.it*

(Received: February 1998; revised: September 1998, January 1999; accepted: July 1999)

This article introduces two new techniques for solving the Quadratic Assignment Problem. The first is a heuristic technique, defined in accordance with the Ant System metaphor, and includes as a distinctive feature the use of a new lower bound at each constructive step. The second is a branch-and-bound exact approach, containing some elements introduced in the Ant algorithm. Computational results prove the effectiveness of both approaches.

The Quadratic Assignment Problem (QAP) is one of the best known and most difficult combinatorial optimization problems, as it is testified by the comparatively small gap existing between the dimension of the problems that can be solved to optimality by means of complete enumeration and the dimension of the problems that can be solved by means of the most advanced exact methods proposed in the literature.

The theoretical and applicative interest of the problem has fostered a large amount of research (Pardalos and Wolkowicz[28] and Çela[18]), which resulted in a number of different approaches, both exact and heuristic.

Exact methods rely on effective lower bounds; the best known of such bounds was proposed by Gilmore[15] and Lawler,[20] but several improvements are available, such as those of Assad and Xu[1] or Carraresi and Malucelli,[7] based on problem reformulations, Rendl and Wolkowicz,[29] based on the eigenvalues of the problem matrices, Resende et al.,[30] based on an interior point algorithm, or Hahn and Grant,[17] based on a dual framework. The most effective exact techniques presented in the literature include those of Mautor and Roucairol,[25] Hahn et al.,[18] and Brungger et al.[3]

However, problem instance of comparatively small dimension cannot yet be solved to optimality. This raises interest in heuristic approaches, among which we recall the simulated annealing of Connolly,[10] the tabu search of Taillard[32] and of Battiti and Tecchiolli,[2] its hybrid with genetic algorithm of Fleurent and Ferland,[14] the GRASP of Li et al.,[21] and the more recent scatter search of Cung et al.[11] Experimental comparisons among them can be found in Taillard[33] and in Maniezzo et al.[23]

The author contributed to the design of a metaheuristic algorithm, the ant system (Colorni et al.,[9] Dorigo et al.[12]), that has also been applied to QAP (Maniezzo et al.,[22] Maniezzo and Colorni[24]). This article describes a substantial improvement of the ant system and its application to QAP. Moreover, it is recognized that the resulting algorithm, ANTS, shares several elements with an approximate branch-and-bound (more specifically, it can be seen as an Approximate Nondeterministic Tree-Search system, hence the name), and the few modifications needed to make it an exact approach are proposed.

The article is structured as follows. Section 1 reviews the mathematical formulation of the QAP and the Gilmore and Lawler bound. Section 2 introduces the ant system and the new adaptations. Section 3 describes how the new general ANTS strategy can be applied to the QAP while Section 4 presents how it can be converted into an exact approach in the case of the QAP. Section 5 shows the computational results obtained by the exact and the heuristic algorithms, finally Section 6 contains some of the conclusions drawn from this work.

1. Mathematical Formulations and Lower Bounds

Several mathematical formulations have been proposed for the QAP, many of which aimed at linearizing the problem; for a recent presentation of the problem, see Çela.[8] The most frequently used formulation is due to Koopmans and Beckmann,[19] and models the case where n facilities are to be assigned to n locations; the assignment cost is due to a direct cost derived from the assignment of a facility to a location and to a cost derived from the amount of *flow* existing between each pair of facilities multiplied by the *distance* between the two locations the facilities are assigned to.

Formally, let $\Phi = \{1, \ldots, n\}$ be an index set of the facilities and $\Lambda = \{1, \ldots, n\}$ be an index set of the locations. Furthermore, let $D = [d_{ih}]$ $i,h = 1, \ldots, n$ be the (possibly asymmetrical) matrix of distances between each pair of locations and let $F = [f_{jk}]$ $j,k = 1, \ldots, n$ be the (possibly asymmetrical) matrix of flows between each pair of facilities. Finally let $C = [c_{ij}]$ $i,j = 1, \ldots, n$ be the cost matrix for the assignment of facilities to locations. A 0/1 binary variable x_{ij} takes value 1 if facility i is assigned to location j, 0 otherwise.

The formulation is as follows.

$$\text{(KB)} \qquad z_{QAP} = min \sum_{i,j=1}^{n} \sum_{h,k=1}^{n} d_{ih} f_{jk} x_{ij} x_{hk} + \sum_{i,j=1}^{n} c_{ij} x_{ij} \qquad (1)$$

Subject classifications: Combinatorial optimization problem.
Other key words: Quadratic assignment problem, the ant system, branch and bound.

subject to the following constraints

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, \ldots, n) \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, \ldots, n) \tag{3}$$

$$x_{ij} \in \{0,1\} \quad (i,j = 1, \ldots, n) \tag{4}$$

Formulation KB can be generalized to consider the case where the assignment cost is given by a generic quadratic 0/1 function. In fact, given a four-dimensional coefficient matrix $\Xi = [\xi_{ijhk}]$, $i,j,h,k = 1, \ldots, n$, QAP asks to solve (GEN)

$$z_{\mathrm{GEN}} = min \sum_{i,j,h,k=1}^{n} \xi_{ijhk}\, x_{ij}\, x_{hk} \tag{5}$$

subject to constraints (2), (3), and (4).

In the following, we will refer to problems presented according to formulation KB, i.e., where the notions of flows and distances can be applied.

The best known lower bound for the QAP was presented by Gilmore[15] and Lawler;[20] it is usually referred to as the Gilmore and Lawler bound (GLB in the following), and has a computational complexity of $O(n^3)$ in the Koopmans and Beckmann case.

Unfortunately, GLB is not very tight for many problems, therefore the research on new lower bounds for the QAP has been and still is very active. For example, Assad and Xu[1] and Carraresi and Malucelli[7] proposed two different QAP reformulation schemes, deriving two bounds that can be computed in $O(n^5)$, Finke, Burkard, and Rendl[13] and Rendl and Wolkowicz[29] proposed two lower bounds based on the eigenvalues of the input matrix. More recently, Resende, Ramakrishnan, and Drezner[30] used an interior point approach to derive their IPLP bound, while Hahn and Grant[17] designed a dual procedure for the general QAP problem.

All these bounds have a higher computational complexity than the GLB for the Koopmans and Beckmann case and, even though significantly improving the GLB, either are not applicable to large instances—as is the case of the interior point bound, which has a complexity of $O(n^7)$—or leave significant gaps on some problem instances, as the QAPLIB (Burkard et al.[6]) testifies.

2. The ANT System

2.1 The Original Algorithm

The ANT system was initially proposed by Colorni, Dorigo, and Maniezzo[9] as a general metaheuristic approach for combinatorial optimization problems. The underlying idea was that of parallellizing search over several constructive computational threads, all based on a dynamic memory structure incorporating information on the effectiveness of previously obtained results.

More specifically, an *ant* is defined to be a simple computational agent, which iteratively constructs a solution for the problem to solve. Partial problem solutions are seen as *states*; each ant *moves* from a state ι to another one ψ, corresponding to a more complete partial solution. At each step σ, each ant k computes a set A_σ^k of feasible expansions to its current state, and moves to one of these in probability.

For ant k, the probability $p_{\iota\psi}^k$ of moving from state ι to state ψ depends on the combination of two values:

1. the *attractiveness* η of the move, as computed by some heuristic indicating the *a priori* desirability of that move. Specifically, a variable $\eta_{\iota\psi}$ quantifies the attractiveness of moving from state ι to state ψ.
2. the *trail level* τ of the move, indicating how proficient it has been in the past to make that particular move: it represents therefore an *a posteriori* indication of the desirability of that move. Specifically, a variable $\tau_{\iota\psi}$ quantifies the past proficiency of moving from state ι to state ψ.

Trails are *updated* at each iteration, promoting those that facilitate moves that were part of "good" solutions. Even though the ant framework can be applied in the general case just described, it is usually the case that the concept of state is relaxed, and a move becomes the insertion in the solution of a variable (j) after having inserted another (i). Thus, evaluating a move means assessing the opportunity of extending a partial solution, where variable i was jsut added, with variable j. Attractiveness, η_{ij}, trail level, τ_{ij}, and move probability p_{ij}^k are accordingly redefined.

The specific formula for defining the probability distribution at each move, proposed in Colorni et al.,[9] makes use of the concept of *tabu list* for ant k, tabu_k, which indicates the set of infeasible moves for ant k. Probabilities were computed as follows:

$$p_{ij}^k = \begin{cases} \dfrac{\tau_{ij}^\alpha + \eta_{ij}^\beta}{\sum_{(ir) \notin tabu_0^k} (\tau_{ir}^\alpha + \cdot \eta_{ir}^\beta)} & \text{if } (ij) \notin \mathrm{tabu}_k \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Exponents α and β are user-defined parameters that specify the relative importance of trail with respect to attractiveness.

After each iteration t, trails are updated following formula (7):

$$\tau_{ij}(t) = \rho \tau_{ij}(t-1) + \Delta \tau_{ij} \tag{7}$$

where ρ is a user-defined coefficient and $\Delta\tau_{ij}$ represents the sum of the contributions of all ants that used move (ij) to construct their solution. The ants' contributions are proportional to the quality of the solutions achieved, i.e., the better an ant solution, the higher will be the trail contributions added to the moves it used.

The ant system simply iterates a main loop where m ants construct in parallel their solutions, thereafter updating the trail levels. Solutions can be carried to the corresponding local optima, for example by means of a simple 2-opt improvement heuristic. The performance of the algorithm depends on the correct tuning of several parameters, namely: α, β, relative importance of trail and attractiveness, ρ, trail persistence, $\tau_{ij}(0)$, initial trail level, m, number of ants, and Q, used for defining to be of high quality solutions with low cost.

2.2 The improved ANTS algorithm

Several improvements and modifications can be suggested to improve the performance of the original ant system algorithm. The ones we advocate in this article are the following.

Use of Lower Bounds (Min Problems)

The attractiveness η of a move can be effectively estimated by means of lower bounds (upper bounds in case of maximization problems) to the cost of the completion of a partial solution. In fact, given a partial problem solution, it is possible to compute a lower bound to the cost of a complete solution containing it. For each feasible move (ij), it is possible to compute the lower bound to the cost of a complete solution containing j: the lower the bound the better the move.

Since a large part of research in combinatorial optimization is devoted to the identification of tight lower bound for the different problems of interest, good lower bounds are usually available. Their use has several advantages, some of which are listed in the following.

- A tight bound gives strong indications on the opportunity of a move.
- When the bound value becomes greater than the current upper bound, it is obvious that the considered move leads to a partial solution that cannot possibly be completed in a solution better than the current best one. The move can therefore be discarded from further analysis.
- If the bound is derived from linear programming and dual cost information is therefore available, it is possible to compute reduced costs for the problem decision variables, which in turn—when compared to an upper bound to the optimal problem solution cost—permit to a priori eliminate some variables. This results in a reduction to the number of possible moves, therefore to a reduction of the search space.
- A further advantage of lower bounds based on linear programming is that the primal values of the decision variables, as appearing in the bound solution, can be used as an estimate of the probability with which each variable will appear in good solutions. This provides an effective way for initializing the trail values, thus eliminating the need for the user-defined parameter $\tau_{ij}(0)$.

Computational Efficiency

An improved computational efficiency is obviously a valuable asset for any heuristic method. While this is usually achieved by a careful implementation and data structure design, some higher-level considerations are possible. For example, formula (6) can be converted to:

$$p_{ij}^{k} = \begin{cases} \dfrac{\alpha \cdot \tau_{ij} + (1-\alpha)\cdot \eta_{ij}}{\sum_{(ir)\notin tabu_k} (\alpha \cdot \tau_{ir} + (1-\eta)\cdot \eta_{ir})} & \text{if } (ij) \notin \text{tabu}_k \\ 0 & \text{otherwise} \end{cases} \tag{6'}$$

Formula (6′) achieves the same objective of formula (6), that of letting the user specify a different relative importance of trail with respect to attractiveness, while eliminating one parameter (β) and using more computationally efficient components, multiplications instead of powers to real-valued exponents.

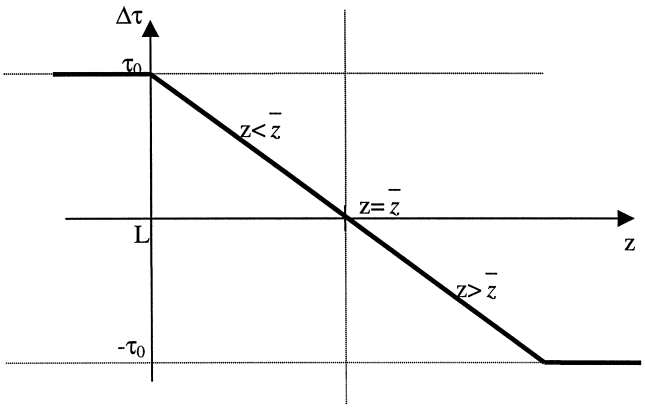


Figure 1. Linear dynamic scaling.

Stagnation Avoidance

Stagnation denotes the undesirable situation in which all ants repeatedly construct the same solutions, making impossible any further exploration in the search process. This derives from an excessive trail level on the moves of one solution, and can be observed in advanced phases of the search process if parameters are not well tuned to the problem. In particular, stagnation derives from a wrong value of parameter ρ in the original algorithm. If it is too high, stagnation might take place, while if it is too low too little information is conveyed from previous solutions and the ant system becomes an iterated randomized greedy procedure. We decided to eliminate this sensitive parameter, by updating trail using a different mechanism. The new procedure evaluates each solution against the last k ones globally constructed by the ANTS algorithm. As soon as k solutions are available, we compute their moving average $\bar{z}$; each new solution z_{curr} is compared to $\bar{z}$ (and then used to compute the new moving average value). If z_{curr} is lower than $\bar{z}$ the trail level of the last solution's moves is increased, otherwise it is decreased. Formula (8) specifies how this is implemented, when coupled with formula (7) above.

$$\Delta\tau_{ij} = \tau_0 \cdot \left(1 - \frac{z_{curr} - LB}{\bar{z} - LB}\right) \tag{8}$$

$\bar{z}$ is the average of the last k solutions and LB is a lower bound to the optimal problem solution cost. Figure 1 depicts the moving average linear scaling function we propose for trail updating.

The use of a dynamic scaling procedure permits to discriminate small achievement in the latest stage of search, while avoiding to focalize search only around good achievement in the earliest stages. The mechanism presented makes no use of parameter ρ and, consequently, of parameter Q. However, a new parameter is introduced, k the cardinality of the moving average, but its value is much less critical than that of the parameters we eliminated.

Using the three modifications proposed, the ANTS system metaheuristic becomes the following.

Improved ANTS algorithm

1. Compute a (linear) lower bound LB to the problem to solve
Initialize τ_{ij} ($\forall i, j$) with the primal variable values
2. **For** $k = 1, m$ (m = number of ants) **do**
repeat
2.1 compute η_{ij} $\forall$ (ij)
2.2 choose the state to move into, with probability given by (6′)
2.3 append the chosen move to the k-th ant's tabu list
until ant k has completed its solution
2.4 carry the solution to its local optimum
end for
3. **For** each ant move (ij),
compute $\Delta\tau_{ij}$ and update the trace matrix by means of (7) and (8)
4. **If not** (end_test) **goto** step 2.

3. ANTS Applied to QAP

Section 2 presented the ANTS metaheuristic, which can be applied to many combinatorial optimization problems. In this Section, we detail how this can be done in the case of a specific problem, namely the QAP. Essentially, this translates into specifying which lower bound has been chosen (step 2.1 of the improved ANTS algorithm) and what is a move in the QAP context (step 2.2), all other elements being those described in Section 2.2.

3.1 Lower Bound

As mentioned in Section 1, there is currently no lower bound for QAP, which is both tight and efficient to compute. A good compromise is the GLB, which can be formulated for the general QAP case; in the Koopmans and Beckmann form it can be formulated as follows. First, for each pair (ij), $i \in \Phi$, $j \in \Lambda$, compute z_{ij}.

$$z_{ij} = min \sum_{h,k=1}^{n} d_{ih} f_{jk} x_{hk} \tag{9}$$

subject to assignment constraints on variables x_{hk}.

Then, GLB is given by

$$\text{(GLBP)} \qquad z_{\text{GLB}} = min\ z = \sum_{i,j=1}^{n} (z_{ij} + c_{ij})\ x_{ij} \tag{10}$$

subject to (2), (3), and (4).

It is well known that the z_{ij} coefficients can be easily computed by preordering the d_{ih} and f_{jk} coefficient vectors in ascending and descending order, respectively, and then computing their scalar product. This reduces the computational complexity of the GLB computation to $O(n^3)$.

Unfortunately, this bound is on the average quite far from the optimal solution. We decided therefore to further trade effectiveness for efficiency, and we used in our procedure a bound, called LBD, worse than GLB but very easy to compute.

LBD has been designed on the basis of the following observations. When computing GLB for a problem, along with the value of the bound one gets the values of the dual variables corresponding to constraints (2) and (3). Let $u_i(0)$, $i = 1, \ldots, n$, be the values of the dual variables corresponding to constraints (2) and let $v_i(0)$, $i = 1, \ldots, n$, the values of the dual variables corresponding to constraints (3): obviously GLB = $\Sigma_{i\in\Phi}\, u_i(0) + \Sigma_{j\in\Lambda}\, v_j(0)$.

The index set Φ can be partitioned into two subsets Φ_1 and Φ_2, denoting the indices of the already assigned facilities and the indices of the still unassigned facilities, respectively. Similarly, the index set Λ can be partitioned into two subsets Λ_1 and Λ_2, denoting the indices of the already assigned locations and the indices of the still unassigned locations, respectively. When a partial solution $\{\Lambda_1, \Phi_1\}$ is considered (assuming for simplicity $c_{ij} = 0\ \forall i,j$) the objective function (1) can be rewritten as:

$$z_{\text{QAP}} = min \sum_{i,h\in\Phi_1} \sum_{j,k\in\Lambda_1} d_{ih} f_{jk} x_{ij} x_{hk} + \sum_{i,h\in\Phi_1} \sum_{j,k\in\Lambda_2} d_{ih} f_{jk} x_{ij} x_{hk} + \sum_{i,h\in\Phi_2} \sum_{j,k\in\Lambda_1} d_{ih} f_{jk} x_{ij} x_{hk} + \sum_{i,h\in\Phi_2} \sum_{j,k\in\Lambda_2} d_{ih} f_{jk} x_{ij} x_{hk}. \tag{1′}$$

A valid lower bound to z_{QAP}, *under the assumption that all* d_{ih} and f_{jk} are nonnegative values, $i, j, h, k = 1, \ldots, n$, is given by

$$z_{\text{GLB}'} = min \sum_{i,h\in\Phi_1} \sum_{j,k\in\Lambda_1} d_{ih} f_{jk} x_{ij} x_{hk} + \sum_{i,h\in\Phi_2} \sum_{j,k\in\Lambda_2} d_{ih} f_{jk} x_{ij} x_{hk}, \tag{11}$$

where the problem has been relaxed and decomposed into two smaller QAPs, the first one with a solution already defined, the second one still completely unsolved.

It is easy to see that

$$z_{\text{LBD}} = \sum_{i,h\in\Phi_1} \sum_{j,k\in\Lambda_1} d_{ih} f_{jk} x_{ij} x_{hk} + \max\left\{ \sum_{i\in\Phi_2} u_i(0) + \sum_{j\in\Lambda_2} v_j(0);\ 0 \right\} \leqslant z_{\text{GLB}'}. \tag{12}$$

The bound obtained is weaker than GLB, but its computational complexity is $O(n)$, much less than that of any other bound so far proposed for QAP. We use bound z_{LBD} computed for a partial assignment $\{\Lambda_1, \Phi_1\}$ as the attractiveness η for moving to the state corresponding to that solution.

3.2 Moves: Assignment Order

A move corresponds to the assignment of a facility to a location, thus adding a new component to the partial solution corresponding to the state from which the move originated. Set Λ is assumed to be pre-ordered, so that an ant beginning to construct a new solution will first assign a facility to the location whose index is the first element of Λ, then one to the second location and so on. Each state ι can therefore be represented by the two ordered subsets Φ_1 and Λ_1, where Λ_1 always consists of the first $|\Lambda_1|$ elements of Λ. All moves originating from ι correspond to the feasible

expansions of ι, that is to all possible assignments of facilities to the location of index $|\Lambda_1| + 1$.

The computational efficiency of the ANTS procedure is greatly affected by the pre-ordering that is imposed on set Λ.

In fact, formula (1′) can be seen as the sum of four components, $z_{QAP} = z_1 + z_2 + z_3 + z_4$, where, in the case of a partial assignment, z_1 refers to an already established cost, z_2 to the cost of the interaction among assigned facilities and assigned locations, and z_4 to the cost of the interaction among unassigned facilities and unassigned locations. Therefore z_1 is a known constant, while z_2, z_3, and z_4 can only be estimated by means of lower bounds. The higher the contribution of z_1 the tighter will probably be the bound to the minimum cost of completing the given partial assignment.

Moreover, increasing the contribution of z_1 implies assigning as soon as possible the more demanding locations, in a stage where all possibilities are still available, without incurring in the risk—due to the poor performance of the lower bound—of adding high cost contributions when completing a partial solution, thus discovering of having constructed a poor solution only when the solution gets completed.

The pre-ordering we used is based on variables $v_j(0)$, $j = 1, \ldots, n$: the higher the value of a dual variable associated to a location the higher it is assumed to be impact of the location on the QAP solution cost, thus the earlier we try to assign a facility to that location.

3.3 Moves: Direction of the Assignment

One last consideration is in order, which despite its simplicity proved to have a definite computational impact. Given the two cost matrices of the Koopmans and Beckmann form, it is in fact customary to assign facilities to locations. However, it is also possible to assign locations to facilities, which in our case would imply a pre-ordering of the facilities by decreasing values of $u_i(0)$, $i = 1, \ldots, n$, and a successive assignment of locations to progressively less cost-intensive facilities.

As shown in Section 5, the choice of assigning facilities to locations or vice-versa is critical; unfortunately we have not been able to define a rule that consistently identifies the best assignment direction. Currently, the best performing criterium is a combination of matrix norms, variances, and number of zeroes, but further exploration is needed.

4. Exact Algorithms

The general structure of the ANTS algorithm is closely akin to that of a standard tree-search algorithm. At each stage we have in fact a partial solution that is expanded by branching on all possible offsprings, computing a bound for each of them, possibly expunging dominated ones, and moving to one of them on the basis of lower bound considerations. By simply adding backtracking and eliminating the Montecarlo choice of the node to move to, we revert to a standard branch-and-bound procedure (as mentioned, this is in fact the reason behind the use of the denotation ANTS, for Approximate Nondeterministic Tree-Search).

The resulting tree to be searched is a n-ary tree where each level corresponds to a location (facility) to which a facility (location) has to be assigned. Throughout search, since the objective is here to prove the optimality of a solution and not to quickly find good upper bounds, we used as lower bound the complete Gilmore and Lawler bound for the problem as derived from formula 1′. In particular, at each node of the search tree we computed the contribution z_1, approximated z_4 by means of the GLB and, following Burkard[4] we computed a lower bound to $z_2 + z_3$ by solving a linear assignment on a matrix $[C_{lm}]$, $l \in \Lambda_2$, $m \in \Phi_2$, where

$$C_{lm} = \sum_{i \in \Lambda_1} \sum_{j \in \Phi_1} (d_{il} f_{jm} + d_{li} f_{mj}) \tag{13}$$

Clearly, at the root node only $z_4 \neq 0$, $z_4 = \text{GLB}$, while in any other node the lower bound is given by the sum of the three components described above. Note that at each node, the lower bound to $z_2 + z_3 + z_4$ can be computed by means of a single linear assignment computation, being the cost matrices of identical dimensions and the contributions to the cost independent from one another. This reduces computing times and provides better results then approximating independently z_4 and $z_2 + z_3$.

A second observation (Mautor and Roucairol[25]) is that the dual variables obtained when computing the linear assignment at each node can be used to expunge offsprings of the incumbent node, before generating them. It is in fact well known from linear programming duality that the introduction of a nonbasic variable in a solution results in an increase of objective function value equal to the value of the reduced cost of the introduced variable. On the basis of the reduced cost matrix obtained at the end of the linear assignment, it is therefore possible to a priori discard the assignments that correspond to variables whose reduced costs exceeds the gap between lower bound, as obtained by z_1 added to the optimal linear assignment cost, and current upper bound to the problem.

Using this common framework, we considered two different branching strategies. The first one is a standard depth-first (DF), where the node expanded at each level is the offspring of the incumbent one with least cost lower bound. The second one is a beam-search (BS) strategy where a number of nodes are expanded at the same level before stepping deeper in the search tree. BS differs from breadth-first search in that the nodes considered for expansion at each level of the search tree are not all unexpanded nodes at that level, but a limited cardinality subset of them, where the number of nodes to expand is a parameter. A standard backtracking mechanism provides the means to possibly return to a not fully expanded level and generate other offspring, always under cardinality limits. This approach has been successfully used on the Travelling Salesman with side constraints problem (Mingozzi et al.[26]).

The beam-search strategy permits us to obtain good upper bounds earlier than depth-first, at the cost of maintaining more computationally expensive memory structures. It is therefore dominated by the depth-first strategy when good upper bounds are available from the start, while it is worthwhile in the opposite case. Since we were interested

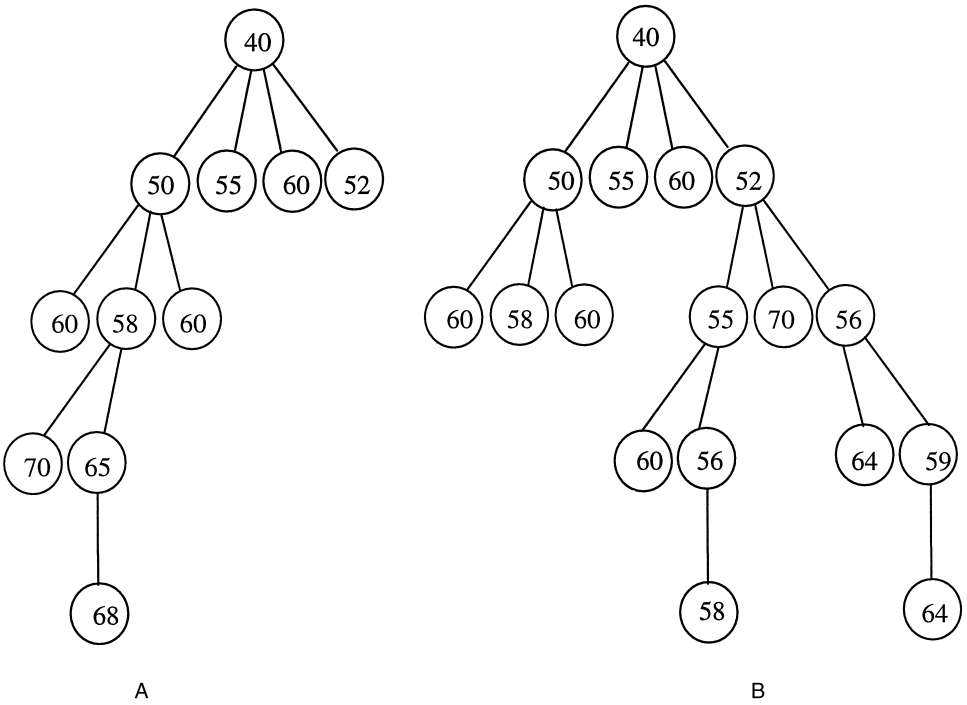


Figure 2. Tree-search visit strategies.

also in evaluating the performance of a truncated branch and bound as a benchmark heuristic, we concentrated on the beam-search strategy.

4.1 Beam Search Exact Algorithm

This algorithm iterates a basic procedure in which a set of promising nodes at a given level of the search tree are expanded to generate their offspring, which all become members of the set of the unexpanded nodes of the lower tree level. A node is considered to be promising in accordance to its lower bound cost. The number of nodes to be expanded at each level is a parameter, which we set to $2n$ after some experimental tuning. When a level has been expanded, two strategies are possible: either expand the level below or expand the level at which there is the unexpanded node of lowest cost lower bound. We decided to proceed downward in order to quickly complete good solutions. In any case, when level $n - 1$ is expanded, we backtrack to the level with the lowest lower bound node, which is usually high in the tree hierarchy.

This process is iterated until all nodes are expanded or until all unexpanded nodes have a lower bound cost that is not smaller than the current upper bound, which is therefore the optimal cost.

The rationale behind this approach lies in the tentative to avoid the possibility of sticking to a branch, which looked promising at the higher levels of the search tree but turns out to be poor considering more complete solutions. Figure 2 compares a depth-first and a beam-search expansion strategies.

Figure 2*A* shows a depth-first visit, while Figure 2*B* shows a beam-search visit, using a degree 2 of parallelism. In the example, thanks to the possibility at level 2 of choosing among 6 nodes (as opposed to the 3 of depth first), it is possible to bypass the offsprings of the node with lower bound of 50, which seemed promising at level 1 but is dominated already at the level immediately below.

The beam-search strategy is therefore useful to find earlier good upper bounds, which in turn permits us to prune more branches than what allowed by depth-first. A second characteristic is that the minimum among the lower cost lower bounds at the different level of the search tree is a lower bound to the whole problem. During the run, each backtrack results in a possible increase of the lower bound to the whole problem, therefore generating a sequence of nondecreasing bounds.

The pseudocode of the tree search algorithm is presented in the following. The general structure is the same as that of a depth-first, the only major difference lying in the structure needed to identify the nodes to be expanded. We used in fact a vector of heaps, one heap for each level of the search tree, so to have an efficient means to obtain the lowest bound node of each level.

More in detail, after having initialized the structures introduced in Section 3.1 (Step 0), the algorithm goes through a main construction loop (Step 1) where the subsequent levels of the search tree are considered. At each level a number of least bound cost nodes is expanded (Step 2) and a bound is computed for each offspring node (Step 3): if undominated the new node is stored for possible expansion. When all levels have been considered, backtracking (Step 4) occurs, in order to expand the most promising still unexpanded nodes, accordingly reinitializing Step 1.

Beam Search Algorithm
step 0: Initialization
read n, D, F;
$z_{ub} = \infty$, $\Lambda_1 = \varnothing$, $\Lambda_2 = \Lambda$, $\Phi_1 = \varnothing$, $\Phi_2 = \Phi$;
compute GLB at node 0 and order the rows of D by decreasing values of the associated dual variables;
step 1: Construction
for each location $i \in \Lambda_2$ **do**
$\Lambda_1 = \Lambda_1 \cup \{i\}$, $\Lambda_2 = \Lambda_2 \setminus \{i\}$;
$lev = i$;
perform step 2
end for
goto step 4
step 2: Branching
for nd = 1 to $2n$ **do**
node = extract_heap (lev);
if lev = n and z_1 (node) $< z_{ub}$ **then**
$z_{ub} = z_1$ (node)
save node
goto step 4
end if
if z_{lb} (node) $< z_{ub}$
Φ_1 = node; $\Phi_2 = \Phi_2$–Φ_1;
for each facility j $\in \Phi_2$ **do**
perform step 3
end for
end if
end for
step 3: Bounding
if not (dominated (lev, j))
compute $z_{lb} = z_1 + z_2 + z_3 + z_4$ with
$\Phi_1 = \Phi_1 \cup j$;
if $z_{lb} < z_{ub}$ insert_heap (lev + 1, $\Phi_1 \cup j$);
end if
step 4: Backtrack
if (empty heap) **then**
print z_{ub} and the optimal solution; **STOP;**
else
{Φ_1, lev} = find_minimum_heap();
$\Phi_2 = \Phi_2$–Φ_1;
redefine Λ_1, Λ_2 according to level lev;
goto step 1;
endif

where *node* is the ordered set of the facilities involved in the partial assignment; *dominated*(i,j) returns *true* if node j at level i is found to be dominated by means of the reduced cost matrix or by the z_{LBD} bound; *insert_heap*(*lev*) inserts into the heap at level *lev* the current node; *extract_heap*(*lev*) extracts the least-cost node from the heap at level *lev*, ordered by increasing lower bound values; and *find_minimum_heap*() returns the level of the search tree at which there is the unexpanded node of least cost lower bound and the node itself.

5. Computational Results

This section presents the results obtained running the Ant System and the Beam Search algorithms. All results have been obtained implementing the algorithms in Fortran 77 and running the codes on a Pentium 166 MHz machine equipped with 32 Mb of RAM. All codes are available from the author.

5.1 ANTS Performance

The ANTS performance depends on the values of three user-defined parameters: m (number of ants), k (width of the moving average windows), and α (relative importance of trail and desirability). We conducted a number of tests over three well-known problem instances to define, in a cœteris paribus framework, which is the best parameter setting for the QAP, which turned out to be $k = 4n$, $m = n/2$ e $\alpha = 0.5$.

Using this configuration, in order to validate the effectiveness of the ANTS approach, we ran 5 times the algorithm over each problem instance of the QAPLIB (Burkard et al.[6]) with $20 \leqslant n \leqslant 40$ plus a problem—MC33—introduced in Maniezzo et al.,[22] allowing 10 minutes of CPU times for each run. For comparison, we used the (publicly available) codes of two of the best heuristics so far presented for the QAP: Robust Tabu Search (Taillard[32]) and GRASP (Li et al.[21]). Moreover, we truncated the execution of our branch and bound after 10 minutes to obtain a further benchmark heuristic.

Table I presents the results so obtained. The columns show: *Problem*, a problem instance identifier, as reported in the QAPLIB; *N*, problem dimension; *Gap GLB*, percentage gap from optimality, or from the best known solution, of the Gilmore-Lawler bound; *TS-best*, percentage gap of the best solution, over five runs, obtained by tabu search; *TS-avg*, percentage gap of the average of the five solutions produced by tabu search; *GRASP-best*, percentage gap of the best solution, over five runs, obtained by GRASP; *GRASP-avg*, percentage gap of the average of the five solutions produced by GRASP; *ANTS-best*, percentage gap of the best solution, over five runs, obtained by the ANTS System; *ANTS-avg*, percentage gap of the average of the five solutions produced by the ANTS System; and *TBB*, percentage gap of the best solution found by the truncated branch and bound procedure. Each figure in columns TS, GRASP, ANTS, and TBB represents a percentage distance from the corresponding best-known solution.

The last row of Table I reports the sum, over all lines above, of the corresponding items. We reported the sum, instead of the average, due to the very good performance that all algorithms have, which lead to results worse than the best known ones only on a comparatively small number of problems. The best performing among the algorithms is ANTS, both considering the best and the average quality of the solutions proposed. We included the column of the Gilmore and Lawler bound both to give an indication of the complexity of the instances and of the strength of the guidance ANTS receives during search. It is interesting to see how, even in the presence of a bad bound at the root node, the nondeterministic strategy followed by ANTS permits to quickly identify good solutions.

We included a column (TBB) reporting the results obtained by a truncated version of the branch-and-bound algorithm described in Section 4. The total gap of TBB on the

Table I. Effectiveness of Heuristic Procedures

			TS		GRASP		ANTS		
Problem	n	Gap GLB	Best	Avg	Best	Avg	Best	Avg	TBB Gap
CHR20A	20	1.92	0.00	0.06	0.18	1.48	0.00	0.00	0.00
CHR20B	20	4.44	0.00	2.06	3.22	4.65	0.00	0.00	0.00
CHR20C	20	39.18	0.00	0.00	0.00	0.00	0.00	0.00	0.00
HAD20	20	10.92	0.00	0.00	0.00	0.00	0.00	0.00	0.00
LIPA20A	20	1.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00
LIPA20B	20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NUG20	20	19.96	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ROU20	20	17.31	0.00	0.00	0.00	0.00	0.00	0.00	0.20
SCR20	20	21.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TAI20A	20	17.46	0.00	0.00	0.00	0.19	0.00	0.00	0.53
TAI20B	20	88.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NUG21	21	24.82	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CHR22A	22	3.77	0.00	0.00	0.55	1.15	0.00	0.00	0.00
CHR22B	22	4.77	0.58	0.71	1.29	2.03	0.00	0.00	0.00
NUG22	22	30.95	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NUG24	24	23.28	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CHR25A	25	27.16	4.06	4.48	2.32	4.64	0.00	0.76	0.00
NUG25	25	23.37	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TAI25A	25	17.55	0.00	0.12	0.40	0.79	0.00	0.00	1.59
TAI25B	25	86.15	0.00	0.00	0.00	0.00	0.00	0.00	1.02
BUR26A	26	4.51	0.00	0.00	0.00	0.00	0.00	0.00	0.21
BUR26B	26	5.20	0.00	0.00	0.00	0.00	0.00	0.00	0.18
BUR26C	26	4.96	0.00	0.00	0.00	0.00	0.00	0.00	0.00
BUR26D	26	5.74	0.00	0.00	0.00	0.00	0.00	0.00	0.00
BUR26E	26	4.38	0.00	0.00	0.00	0.00	0.00	0.00	0.02
BUR26F	26	4.91	0.00	0.00	0.00	0.00	0.00	0.00	0.03
BUR26G	26	7.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00
BUR26H	26	7.97	0.00	0.00	0.00	0.00	0.00	0.00	0.00
KRA30A	30	23.10	0.00	0.00	0.00	0.34	0.00	0.00	3.19
KRA30B	30	24.45	0.00	0.00	0.00	0.15	0.00	0.00	0.83
LIPA30A	30	0.35	0.00	0.00	0.00	0.19	0.00	0.00	0.00
LIPA30B	30	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NUG30	30	25.88	0.00	0.00	0.29	0.39	0.00	0.00	0.72
TAI30A	30	17.24	0.00	0.16	1.19	1.53	0.00	0.13	0.88
TAI30B	30	93.54	0.00	0.00	0.04	0.11	0.00	0.00	11.11
THO30	30	39.59	0.00	0.00	0.00	0.15	0.00	0.00	1.21
ESC32A	32	73.08	0.00	0.00	0.00	2.77	0.00	0.00	7.69
ESC32B	32	42.86	0.00	0.00	0.00	0.00	0.00	0.00	9.52
ESC32C	32	45.48	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ESC32D	32	47.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ESC32E	32	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ESC32F	32	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ESC32G	32	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ESC32H	32	41.32	0.00	0.00	0.00	0.00	0.00	0.00	0.91
MC33	33	38.18	0.00	0.12	0.00	0.07	0.00	0.00	1.30
TAI35A	35	19.44	0.57	0.65	1.61	1.80	0.20	0.32	2.08
TAI35B	35	89.11	0.00	0.00	0.19	0.23	0.00	0.00	4.54
STE36A	36	25.22	0.00	0.00	1.30	1.76	0.00	0.00	3.04
STE36B	36	45.41	0.00	0.00	0.92	1.44	0.00	0.00	3.97
STE36C	36	22.40	0.00	0.00	0.46	0.67	0.00	0.00	0.79
LIPA40A	40	0.35	0.00	0.00	0.97	1.11	0.00	0.00	0.96
LIPA40B	40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TAI40A	40	20.59	0.67	0.93	1.86	2.06	0.07	0.47	1.92
TAI40B	40	92.61	0.00	0.00	0.01	0.03	0.00	0.00	4.80
THO40	40	40.21	0.01	0.06	0.65	0.91	0.00	0.03	2.49
SUM			5.89	9.36	17.46	30.56	0.28	1.71	65.73

Table II. Efficiency of Heuristic Procedures

	TS		GRASP		ANT	
probl	tbest	tavg	tbest	tavg	tbest	tavg
BUR26A	44	46	0	12	3	10
BUR26B	75	149	7	27	3	26
BUR26C	29	49	5	24	8	13
BUR26D	231	234	3	19	3	48
BUR26E	26	81	17	79	0	12
BUR26F	121	172	5	15	3	20
BUR26G	56	116	6	70	0	11
BUR26H	42	86	12	33	0	17
CHR20A	221	-D-	-D-	-D-	17	43
CHR20B	285	-D-	-D-	-D-	237	274
CHR20C	34	235	0	22	1	9
CHR22A	64	171	-D-	-D-	26	48
CHR22B	-D-	-D-	-D-	-D-	53	156
CHR25A	-D-	-D-	-D-	-D-	98	551
ESC32A	203	258	78	-D-	69	165
ESC32B	26	49	3	28	24	52
ESC32C	1	2	0	0	0	2
ESC32D	28	54	0	3	13	21
ESC32E	0	0	0	0	0	0
ESC32F	0	0	0	0	0	0
ESC32G	0	0	0	0	0	0
ESC32H	50	69	2	10	0	20
HAD20	2	2	0	2	2	2
KRA30A	79	103	157	-D-	12	46
KRA30B	48	157	127	-D-	9	25
LIPA20A	1	2	1	2	0	1
LIPA20B	0	1	0	0	0	0
LIPA30A	14	31	2	-D-	1	7
LIPA30B	1	3	0	4	0	0
LIPA40A	69	94	-D-	-D-	4	25
LIPA40B	11	3	1	11	0	0
NUG20	2	2	0	5	0	1
NUG21	5	7	1	55	0	3
NUG22	8	15	6	24	0	4
NUG24	5	12	14	57	1	1
NUG25	4	16	3	53	0	2
NUG30	41	106	-D-	-D-	14	30
ROU20	39	103	19	165	7	9
SCR20	2	9	13	77	1	5
STE36A	224	247	-D-	-D-	52	178
STE36B	77	117	-D-	-D-	18	29
STE36C	140	180	-D-	-D-	48	78
TAI20A	71	93	424	-D-	7	21
TAI20B	7	127	1	6	1	3
TAI25A	9	-D-	-D-	-D-	56	116
TAI25B	24	35	11	83	2	7
TAI30A	435	-D-	-D-	-D-	35	314
TAI30B	145	169	-D-	-D-	24	223
TAI35A	-D-	-D-	-D-	-D-	240	232
TAI35B	119	249	-D-	-D-	20	42
TAI40A	-D-	-D-	-D-	-D-	382	427
TAI40B	203	237	-D-	-D-	58	340
THO30	8	70	69	-D-	7	10
THO40	-D-	-D-	-D-	-D-	507	581
MC33	268	-D-	12	-D-	96	186

whole set of problems is definitely greater than that of any of the other three algorithms, however one can notice that on problems up to dimension 30 (Table I is organized by increasing problem dimensions), TBB produces solutions of a quality comparable with those of the others, while greater dimensions require more than the allotted 10 minutes CPU time.

Table II presents the CPU times needed by the three metaheuristics to produce their best solutions overall (tbest) and the average of the CPU times used in the five runs on the same problem to obtain the best solution of the run (tavg). We did not present the results of the truncated branch and bound, being dominated by those of the other heuristics, and we shadowed the cells corresponding to solutions of worse quality that that found by some other algorithm, thus the reported times indicate the CPU time needed to produce results of the same quality. Here again it is possible to see how ANTS was able to produce its results in a time on the average not longer than that required by the other algorithms.

5.2 Exact Methods

This section reports about the results obtained by the BS algorithm described in Section 4. We carried out two sets of tests because we found to be of decisive importance the order of the assignment (facilities to locations or vice-versa), but we could not produce a rule that consistently suggests the best order: as mentioned in Section 4, the better one we found, used in procedure TBB of Section 5.1, is a combination of considerations based on the norms of the D and F matrices, of their variances and of their percentage number of zeroes, but surely better rules can be designed.

Table III reports the results obtained on all instances of the QAPLIB with dimension less or equal to 20, and for some instance of greater dimension that we have been able to solve (we did not try problems with $n > 40$). We report under DF the results obtained assigning facilities to locations, and under FD those obtained assigning locations to facilities. The columns show: Problem, a problem instance identifier, as reported in the QAPLIB; n, dimension of the problem; Ttot, CPU time needed to solve the problem, in seconds; and Nodes, number of nodes used by the branch and bound procedure. Bold/italic cells specify which of the two approaches produced better results. In case an algorithm could not complete a search (i.e., it could not find an optimal solution or it could not prove its optimality), we report n.a.

One can see how, in general, the DF strategy is more proficient; however, there are cases (see ELS19 or TAI12B) where the reverse approach is by far better. We have been unable to prove the optimality of only three problems with $n \leqslant 20$, namely ESC16B, TAI20A, and ROU20. We terminated the search on them because an interpolation of the rate of increase of the lower bound suggested that it would have taken more than a month to complete the search on our PC, which was more than we could wait. On the other hand, we tested several problems with $20 < n \leqslant 40$, solving seven of them (for TAI25B we were the first to be able to prove the optimality of the solution).

Table III. Comparison of the Branch-and-Bound Procedures

		DF		FD				DF		FD	
Problem	n	Ttot	Nodes	Ttot	Nodes	Problem	n	Ttot	Nodes	Ttot	Nodes
CHR12A	12	0	***75***	0	1133	ESC16J	16	1742	***8951500***	n.a.	
CHR12B	12	0	***51***	0	1800	HAD16	16	255	***680928***	n.a.	
CHR12C	12	0	***512***	0	1894	NUG17	17	2866	***8110410***	8578	25155640
HAD12	12	1	***2713***	11	70548	TAI17A	17	34217	***88711939***	16279	40071213
NUG12	12	2	***15852***	3	15452	CHR18A	18	4	***6261***	410	1456827
ROU12	12	4	23336	3	***16781***	CHR18B	18	1	***1586***	3	9373
SCR12	12	0	***866***	1	9255	HAD18	18	12887	***2335316***	n.a.	
TAI12A	12	1	6618	1	***2466***	NUG18	18	12621	***37396822***	63481	137456605
TAI12B	12	175	1502452	10	***51663***	ELS19	19	n.a.		0	**398**
HAD14	14	4	***12512***	342	1740403	CHR20A	20	1	***1231***	25	68813
NUG14	14	30	***124143***	105	437290	CHR20B	20	102	193367	28	**85001**
CHR15A	15	0	***1333***	8	47332	CHR20C	20	0	***490***	220	6544968
CHR15B	15	0	***278***	13	72116	HAD20	20	68344	***81053458***	n.a.	
CHR15C	15	2	6040	1	***4451***	LIPA20A	20	1	***629***	27	24533
NUG15	15	77	***280220***	277	1027773	LIPA20B	20	0	***24***	0	19
ROU15	15	509	1676099	303	***933419***	NUG20	20	388800	***724187189***	n.a.	
SCR15	15	1	***1334***	24	97236	ROU20	20	n.a.		n.a.	
TAI15A	15	1435	5115875	7394	***30172938***	SCR20	20	521	***909592***	n.a.	
TAI15B	15	39	***140841***	n.a.		TAI20A	20	n.a.		n.a.	
ESC16A	16	69	***272254***	n.a.		TAI20B	20	6553	***14689318***	n.a.	
ESC16B	16	n.a.		n.a.		CHR22A	22	2	***1942***	15467	26319756
ESC16C	16	28260	***182642560***	n.a.		CHR22B	22	7	***9904***	2860	4858344
ESC16D	16	27267	***163709710***	n.a.		CHR25A	25	75	***59986***	n.a.	
ESC16E	16	15	***53113***	n.a.		TAI25B	25	561600	***613059891***	n.a.	
ESC16F	16	1	3520	1	3520	LIPA30A	30	51	***13623***	n.a.	
ESC16G	16	2	***7952***	n.a.		LIPA30B	30	0	***61***	n.a.	
ESC16I	16	140	***693570***	n.a.		LIPA40B	40	1	***80***	n.a.	

Table IV. Comparison of Different Exact Algorithms

Algorithm	nug08	nug12	nug15
Burkard-Derigs[5]	403	36966	2064415
Roucairol[31]	428	83379	
Pardalos-Crouse[27]	798	42706	1596353
Mautor-Roucairol[25]	32	3474	97287
Hahn, Grant, and Hall[18]	131	380	2203
Brungger et al.[3]	32	3359	49063
Maniezzo and Colorni,[24] only GL	810	46003	1679999
Maniezzo and Colorni,[24] GL + z_{LBD} dominance rule	511	28212	957520
BS, 1998	314	15852	280220

It is not easy to compare BS with other exact algorithms, as the number of expanded nodes is not a widely available datum, while the computer used vary from a PC to such machines as a Cray 2 or a Cyber 76, making it difficult to compare CPU times.

We report in Table IV the results we have been able to find, relative to the number of expanded nodes of nine algorithms on three Nugent instances. The italic font lines report of algorithms that make use of dominance rule specific for problems structured as the Nugent's, which we did not include in our procedure to keep it as general as possible. If we refer to branching algorithms using the same dominances, essentially only lower and upper bounds, it is possible to see how BS expands less nodes than the others do.

To evaluate the effectiveness of the different implementational elements that we introduced in Section 4, we recorded for three problem instances (Nug08, Nug12, and Nug15) the number of nodes needed to solve them with and without the considered component. The last three lines of Table IV present the results obtained using only the Gilmore and Lawler bound, computed by means of a single assignment for z_2, z_3, and z_4, those obtained adding the dominance rule based on z_{LBD} and on the reduced costs, and finally those obtained by the full BS procedure, making also use of the ordering derived from the dual variable values. It is apparent how each component contributes effectively in reducing the search space.

We noticed that the effectiveness of the reduction based on the dominance criteria does not vary much with the problem types, consistently yielding a gain factor ranging from 1.5 to 3. The ordering has on the contrary an impact

that depends much on the problem type. We tested on six instances (SCR15, CHR18A, HAD14, TAI10B, LIPA20A, and ESC16G) the gain factor induced by the use of the ordering only: it varies from 3.88, for problem CHR18A, to 52.4, for problem SCR15. In all problem tested, however, we always witnessed a significant decrease in the number of nodes.

A further topic we want to briefly discuss is the lower bound to the problem and its impact on search. As outlined in Section 3, we have a high number of nodes in the higher levels of the search tree: this is due to the fact that the lower bound used (Gilmore and Lawler, as in Eq. 1′) is not tight, therefore very few nodes can be expunged at the higher levels of the tree. As soon as its different components add to a value closer to the current upper bound, many branches are cut and few nodes remain to be explored. For example, recording the number of nodes expanded at each level of the search tree for problem NUG14, we have the maximum number of nodes (46592) at level 5, then a monotonic decrease (level 6, 35357; level 7, 18006; level 8, 4964) until it goes under 1000 nodes at level 9 (704) and it becomes exactly 0 at level 13, where it is proved that no node can yield better solutions than the current upper bound.

Finally, we want to point out that usually, with the problem dimensions up to 20 as in our test, the optimal solution value is found very quickly, as testified by Table I. Most of the CPU effort is spent in proving the optimality of the solution found. However on some problems, such as TAI12B, we witness a slower reciprocal convergence of upper and lower bound.

6. Conclusions

In this article we described a development of a metaheuristic algorithm, ANTS, originally designed after a biological metaphor. The metaheuristic, now efficient on combinatorial optimization problems, shares several elements with other approaches so far presented (most noticeably Scatter Search, Glover,[16] GRASP, Li et al.,[21] and approximate branch and bounds) embedding them in an innovative framework.

We tested the algorithm on a specific hard combinatorial optimization problem, the Quadratic Assignment, both because of its inherent difficulty, thus providing a reliable benchmark, and because we tested on it also a former version of the ant system (Maniezzo et al.[22, 24]). The computational results testify the effectiveness of the new approach, both when compared to the other metaheuristics and to the old ant approach.

Moreover, being the redesigned ANTS system close to a branch-and-bound strategy, we adapted it in that direction, obtaining an exact algorithm. We introduced some simple considerations that, when implemented, allowed the branch and bound to obtain on a PC results comparable to those of the state-of-the-art exact procedures, in one case allowing for the first time to solve to optimality a standard problem instance.

Several improvements can be made to both the exact and the approximate version of the tree search; first and foremost, we believe that all results presented here can be significantly improved by using a tighter lower bound than the Gilmore and Lawler's, such as the bound proposed by Hahn and Grant or that of Carraresi and Malucelli.

Acknowledgments

I am grateful to A. Mingozzi for the helpful discussions on the beam search strategy, and to M. Giovannetti for the careful implementation of the relative code.

References

1. A.A. Assad and W. Xu, 1985. On Lower Bounds for a Class of Quadratic {0,1} Programs, *Operations Research Letters 4*, 175–180.
2. R. Battiti and G. Tecchiolli, 1994. The Reactive Tabu Search. *ORSA Journal on Computing 6*, 126–140.
3. A. Brungger, J. Clausen, A. Marzetta, and M. Perregaard, 1996. Joining Forces in Solving Large-Scale Quadratic Assignment Problems in Parallel, *DIKU Technical Report TR-96-23*, University of Copenhagen.
4. R.E. Burkard, 1984. Quadratic Assignment Problems, *European Journal of Operational Research 15*, 283–289.
5. R.E. Burkard and U. Derigs, 1980. Assignment and Matching Problems: Solution Methods with Fortran Programs, *Lecture Notes in Economics and Mathematical Systems*, Vol. 184, Springer, Berlin.
6. R.E. Burkard, S.E. Karisch, and F. Rendl, 1994. QAPLIB—A Quadratic Assignment Problem Library, *Technical Report n.287*, Technische Universität Graz.
7. P. Carraresi and F. Malucelli, 1992. A New Lower Bound for the Quadratic Assignment Problem, *Operations Research 40*, Suppl. 1, S22–S27.
8. E. Çela, 1998. *The Quadratic Assignment Problem*. Kluwer Academic Publishers, Boston.
9. A. Colorni, M. Dorigo, and V. Maniezzo, 1991. Distributed Optimization by Ant Colonies, *Proceedings of ECAL91—European Conference on Artificial Life*, Paris, France, Elsevier Publishing, Amsterdam, pp. 134–142.
10. D.T. Connolly, 1990. An Improved Annealing Scheme for the QAP, *European Journal of Operational Research 46*, 93–100.
11. V.-D. Cung, T. Mautor, P. Michelon, and A. Tavares, 1997. A Scatter Search Based Approach for the Quadratic Assignment Problem. *Proceedings of the IEEE-ICEC'97 Conference*, Indianapolis.
12. M. Dorigo, V. Maniezzo, and A. Colorni, 1996. Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 26*, 29–41.
13. G. Finke, R.E. Burkard, and F. Rendl, 1987. Quadratic Assignment Problems, *Annals of Discrete Mathematics 31*, 61–82.
14. C. Fleurent and J.A. Ferland, 1994. Genetic Hybrids for the Quadratic Assignment Problem, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 16*, (P.M. Pardalos and H. Wolkowicz, eds.), 173–188.
15. P. Gilmore, 1962. Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem, *Journal of SIAM 10*, 305–313.
16. F. Glover, 1995. Scatter Search and Star-Paths: Beyond the Genetic Metaphor, *OR Spectrum 17*, 125–137.
17. P. Hahn and T. Grant, 1998. Lower Bounds for the Quadratic Assignment Problem Based upon a Dual Formulation, *Operations Research 46*, 912–922.
18. P. Hahn, T. Grant, and N. Hall, 1998. A Branch and Bound Algorithm for the Quadratic Assignment Problem Based on the Hungarian Method, *European Journal of Operational Research 108*, 629–640.

19. T.C. KOOPMANS and M.J. BECKMANN, 1957. Assignment Problems and the Location of Economic Activities, *Econometrica 25*, 53–76.
20. E. LAWLER, 1963. The Quadratic Assignment Problem, *Management Science 9*, 586–599.
21. Y. LI, P.M. PARDALOS, and M.G.C. RESENDE, 1994. A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment and Related Problem. In P.M. Pardalos and H. Wolkowicz (eds), *Quadratic Assignment and Related Problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 16*, 237–261.
22. V. MANIEZZO, A. COLORNI, and M. DORIGO, 1994. The Ant System Applied to the Quadratic Assignment Problem. *Technical Report IRIDIA/94-28*, Université Libre de Bruxelles, Belgium.
23. V. MANIEZZO, A. COLORNI, and M. DORIGO, 1995. Algodesk: An Experimental Comparison of Eight Evolutionary Heuristics Applied to the Quadratic Assignment Problem. *European Journal of Operational Research 81*, 188–205.
24. V. MANIEZZO and A. COLORNI, 1998. The Ant System Applied to the Quadratic Assignment Problem, *IEEE Transactions on Knowledge and Data Engineering*, to appear.
25. T. MAUTOR and C. ROUCAIROL, 1993. A New Exact Algorithm for the Solution of Quadratic Assignment Problems, *Discrete Applied Mathematics 55*, 281–293.
26. A. MINGOZZI, L. BIANCO, and S. RICCIARDELLI, 1997. Dynamic Programming Strategies for the Traveling Salesman Problem with Time Window and Precedence Contraints, *Operations Research 45*, 365–377.
27. P.M. PARDALOS and J. CROUSE, 1989. A Parallel Algorithm for the Quadratic Assignment Problem, *Proceedings of Supercomputing '89, ACM*, 351–360.
28. P.M. PARDALOS and H. WOLKOWICZ (eds.), 1994. Quadratic Assignment and Related Problems, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 16*.
29. F. RENDL and H. WOLKOWICZ, 1992. Applications of Parametric Programming and Eigenvalue Maximization to the Quadratic Assignment Problem, *Mathematical Programming 53*, 63–78.
30. M.G.C. RESENDE, K.G. RAMAKRISHAN and Z. DREZNER, 1995. Computing Lower Bounds for the Quadratic Assignment Problem with an Interior Point Algorithm for Linear Programming, *Operations Research 43*, 781–791.
31. C. ROUCAIROL, 1987. A Parallel Branch and Bound Algorithm for the Quadratic Assignment Problem, *Discrete Applied Mathematics 18*, 211–225.
32. E. TAILLARD, 1991. Robust Taboo Search for the Quadratic Assignment Problem, *Parallel Computing 17*, 443–455.
33. E. TAILLARD, 1995. Comparison of Iterative Searches for the Quadratic Assignment Problem, *Location Science 3*, 87–105.