

A STUDY OF QUADRATIC ASSIGNMENT PROBLEM INSTANCES THAT ARE DIFFICULT FOR META-HEURISTIC METHODS

ZVI DREZNER¹

California State University-Fullerton, zdrezner@exchange.fullerton.edu

PETER M. HAHN²

The University of Pennsylvania, hahn@seas.upenn.edu

ÉRIC D. TAILLARD

The University of Applied Sciences of Western Switzerland, eric.taillard@eivd.ch

Abstract.

The quadratic assignment instances frequently used in the literature are relatively well solved by heuristic approaches. Indeed, solutions at a fraction of one percent from the best known solution values are rapidly found by most heuristic methods. Exact methods are not able to prove optimality for these instances as soon as the problem size approaches 30 to 40. This article presents new QAP instances that are ill conditioned for many metaheuristic-based methods. However, these new instances are shown to be solved relatively well by some exact methods since problem instances up to a size of 75 have been exactly solved.

Key words: Quadratic assignment problem, local search, branch & bound, benchmarks.

1. Introduction.

1.1 The quadratic assignment problem (QAP).

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

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

We denote by $\Pi(n)$ the set of permutations of n elements. Shani and Gonzalez (1976) showed that the QAP is NP -hard and that there is no ε -approximation polynomial algorithm for the QAP unless $P = NP$.

One of the oldest applications of the QAP is the placement of electronic modules (Steinberg 1961, Nugent et al., 1968). In this case, a_{ij} is the number of connections between two electronic modules and b_{ij} is

¹ The authors appear in alphabetical order.

² Corresponding author.

the distance between locations i and j on which modules can be placed. By solving a QAP, one tries to minimize the total length of the electrical connections.

Another application is the assignment of specialized rooms in a building (Elshafei, 1977). In this case, a_{ij} is the flow of people that must go from service i to service j and b_{ij} is the time for going from room i to room j . A more recent application is the assignment of gates to airplanes in an airport; in this case, a_{ij} is the number of passengers going from airplane i to airplane j (a special “airplane” is the main entrance of the airport) and b_{ij} is the walking distance between gates i and j .

Finally, let us mention other applications in imagery (Taillard, 1995), turbine runner balancing (Laporte and Mercure, 1988) and the fact that problems such as the traveling salesman or the linear ordering can be formulated as special QAPs.

1.2. Short literature review.

The QAPLIB library (Burkard, Çela, Karisch and Rendl, 1997) available on the web contains about 140 different problem instances that are frequently used by researchers for comparing methods. The size of these instances ranges from 12 to 256, but the size of only 4 instances is larger than 100. Most of these instances are relatively well solved by heuristic methods (for instance, a 10 years old tabu search (Taillard 91) coded in a few dozen lines): almost all metaheuristic-based methods are able to find solutions at a fraction of one percent above the best known solutions. Exact methods have difficulties in proving optimality of the best solutions known for the problem instances contained in QAPLIB. Some instances of size lower than 30 are still open and none of the instances of size 40 or higher are exactly solved (except those with optimum known by construction).

The aim of this study is to propose new problem instances whose characteristics are complementary to QAPLIB ones, namely, they are relatively well solved by exact methods but ill conditioned for some local search methods.

For researchers working on exact methods, these new instances are interesting since they have been found to be solvable for sizes larger than 70. The main difficulty in dealing with such instances is the memory needed to store data structures. This kind of problem does not often arise with QAPLIB instances, since instances of size 40 cannot yet be solved exactly. In fact, there is a QAPLIB instance of size 25 (the Tai25a) that to date is not solved. It may, however, be solved in the next few months.

For researchers working on heuristic methods, these new instances are interesting for several reasons. First, most recent heuristic methods are built on neighborhood search based on transpositions (pair exchanges). This neighborhood concept is not effective for our new instances. Indeed, there are local optima with a value of the objective function more than 3000% above the optimum. Therefore, a new neighborhood concept must be used. Second, the variance of the solutions produced by heuristic methods can be very high. It is not judicious to proceed to statistical comparisons by considering only average and standard deviation of the solution values produced by different methods, as is often the case in the literature. Therefore, new

comparison techniques have to be constructed. Third, it is necessary to find faster heuristic methods, since the size of some of these new instances is quite large (up to 729).

The quadratic assignment problem was one of the first problems solved by metaheuristic methods first conceived in the 1980's. Burkard and Rendl (1984) proposed a simulated annealing procedure that was able to find much better solutions than all the previously designed heuristic methods. Six years later, Connolly (1990) proposed an improved annealing scheme. His method is easy to set up, since the user has to select the number of iterations; and all other parameters are automatically computed. The code for this method is available on the Internet at <http://www.eivd.ch/ina/collaborateurs/etd/default.html>. Thus, it was convenient for us to include Connolly's method in our computational results. At around the same time, Skorin-Kapov (1990) proposed a tabu search. Then, Taillard (1991) proposed a more robust tabu search, with fewer parameters and running n times faster than the previous implementation. Even though Taillard's method was proposed over 11 years ago it remains one of the most efficient for some problem instances. It is available on the web (see address above). Other tabu searches have been proposed, such as the reactive tabu search of Battiti and Tecchiolli (1994) and the star-shape diversification approach of Sondergeld and Voss (1996).

Genetic algorithms have also been proposed, for instance by Tate and Smith (1995), but hybrid approaches, such as those of Fleurent and Ferland (1994), Drezner (2002b, 2002c) are more efficient. More recently, ant system approaches (Gambardella et al., 1999, Stützle and Hoos 1999) have been proposed, as well as a scatter search (Cung et al., 1997). Some of these methods have been compared in Taillard (1995) who showed that the efficiency of these methods strongly depends on the problem instance type to which they are applied.

2. New difficult instances.

2.1 The Drex Instances

These problem instances are available in <http://business.fullerton.edu/zdrezner>. They are generated by the following principles.

1. All problems are symmetric.
2. A configuration of k by l squares is the base for the problem.
3. The number of facilities is $n=k \times l$.
4. Each square has between two and four "adjacent" squares.
5. A random permutation (the optimal solution) is generated and facilities are "assigned" to the appropriate square by the permutation.
6. Most pairs of facilities get a zero weight (i.e., flow between them). Weight of flow is bi-directional. Only adjacent facilities get a randomly generated positive weight between one and ten.

7. Distances are generated such that all adjacent cells get a distance of one, and all other pairs of cells get a distance randomly generated between two and ten.

Note that:

1. The value of the objective function for the permutation is the sum of all the weights because for the permutation all positive weights are multiplied by a distance of "1".
2. The sum of the weights is also a lower bound because all distances are ≥ 1 .
3. The permutation and three other mirror images for $k \neq 1$ are the only optimal solutions.

2.2 The Taixxeey instances

To build instances that are difficult to solve by most of the heuristic method available, we observed that most solution methods are based on a transposition neighborhood (exchange of two facilities). Therefore, we first searched for a problem structure for which local optima, for this kind of neighborhood, are relatively far apart. Second, we wanted to be aware of very good solutions to the new instances, without necessarily knowing the global optimum, so that these problems would remain attractive. Third, we wanted to generate large problem instances. Finally, we wanted the difficulty of the instances to grow with size, especially for heuristic method working more globally than local searches, such as genetic (hybrid) algorithms.

To meet this last condition, we observed that genetic algorithms, as well as exact methods, generally have difficulties in finding the optimum of problem instances for which distances and flows are uniformly distributed (the instance Tai25a, of size $n=25$ was uniformly generated). However, such instances are relatively well approached by heuristic methods (solutions at a fraction of a percent above the best known solution are found very easily). Therefore, we propose to recursively build our new instances. For generating a QAP instance of size $st \times st$, the idea is to uniformly generate t flows and distances matrices of size $s \times s$. The t flows matrices are placed in a block-diagonal matrix of size $st \times st$, the elements outside the blocks being set to 0 and the t distances matrices are used to create a block-diagonal matrix but with elements outside the blocks being set to infinity. These new $st \times st$ QAP instances can be solved as follows: First, consider the t^2 QAP instances of size $s \times s$ that can be obtained by combining each $s \times s$ flows matrix and each $s \times s$ distances matrix. These t^2 instances are optimally solved, and the optimum solution values are stored in a $t \times t$ matrix C . The optimal solution value of the $st \times st$ QAP instance can be obtained by solving the linear assignment problem on the C matrix.

In order to somewhat hide the block-diagonal structure of the $st \times st$ instance, few elements outside the diagonal of the flows matrix are set to a small positive value and the elements outside the diagonal of the distances matrix are set to relatively large values, but not infinity. By doing that, the optimality property of the linear assignment problem on the C matrix is lost. Naturally, the process can be iterated and u $st \times st$ instances can be generated to create a new $stu \times stu$ instance. The Taixxeey problem instances have being so created (xx stands for the size of the instance and yy for the number of the instance). 20 instances of sizes $n=27$ ($=stu=3 \cdot 3 \cdot 3$), $n=45$ ($=5 \cdot 3 \cdot 3$), $n=75$ ($=5 \cdot 5 \cdot 3$), $n=125$ ($=5 \cdot 5 \cdot 5$), $n=175$ ($=7 \cdot 5 \cdot 5$), $n=343$ ($=7 \cdot 7 \cdot 7$) and 10

instances of size $n=729$ ($=9 \cdot 9 \cdot 9$) are available on the web page: <http://www.eivd.ch/ina/collaborateurs/etd/problemes.dir/qap.dir/qap.html>.

Since these instances embed uniformly generated instances, they are asymptotically difficult to solve exactly. For the case of a transposition neighborhood, one must move through s very bad solutions to make even a small change in the structure of a solution (exchanging two primitive blocks of size s) and st moves are required to exchange two secondary blocks. As explained below, tabu search or simulated annealing based methods become inefficient using such a structure.

3. Heuristic techniques used.

3.1 The Hybrid Genetic Algorithm

A genetic algorithm produces offspring by mating parents and attempting to improve the population make-up by replacing existing population members with superior offspring. A hybrid genetic algorithm, sometimes called a memetic algorithm (Moscato, 2002), incorporates some improvement heuristic (a post merging procedure) on every offspring before considering its inclusion into the population. The specific hybrid genetic algorithm tested in this paper is presented in Figure 1:

- 1 A starting population is randomly selected, and the post-merging procedure is applied on each starting population member.
- 2 The following is repeated for a pre-specified number of generations:
 - 2a Two population members are randomly selected (each population member has an equal chance of being selected, regardless of its objective function value) and merged to produce an offspring.
 - 2b The post merging procedure is applied on the merged solution, possibly improving it.
 - 2c If the value of the objective function for the offspring is not better than the value of the objective function for the worst population member, the offspring is ignored and the process of the next generation starts.
 - 2d Otherwise,
 - If the offspring is identical to an existing population member, it is ignored and the process of the next generation starts.
 - If the offspring is different from all population members, the offspring replaces the worst population member.

Figure 1: General scheme of the hybrid genetic algorithm

For the implementation of the hybrid genetic algorithm for the solution of the quadratic assignment problem, each solution is defined by the list of facilities assigned to sites $1, \dots, n$. We summarize the merging and post-merging procedures used in this paper. These are described in detail in Drezner (2002b, 2002c).

3.1.1 The Merging Procedure

The merging procedure used is the “cohesive merging procedure” (Drezner, 2002b) and is presented in Figure 2:

- 1 Repeat the following n times, once for each site (called the pivot site).
 - 1a The median distance from the pivot site to all sites is calculated.
 - 1b Sites that are closer than the median to the pivot site are assigned the facility from the first parent.
 - 1c All other sites are assigned a facility from the second parent.
 - 1d A list of unassigned facilities is created.
 - 1e All facilities from the second parent that are assigned twice are replaced with an unassigned facility.
 - 1f The value of the objective function for the merged solution is calculated.
- 2 The best of the n merged permutations (i.e., with the best value of the objective function) is selected for a post merging procedure.
- 3 The result of the post merging procedure is the offspring.

Figure.2: Cross-over operator used for merging two solutions.

3.1.2 The Post Merging Procedure

The “short” concentric Tabu search is used here. A concentric Tabu search was first presented in Drezner (2002a) and was used as a post merging procedure in hybrid genetic algorithms (Drezner 2002b, 2002c). The short version is proposed in Drezner (2002c).

A single iteration of the concentric Tabu search is similar to the “variable neighborhood search” of Mladenovic and Hansen (1997) or Hansen and Mladenovic (2001). The search is performed in “rings” around the center solution, proceeding from one ring to a larger one, and so on, until the pre-specified depth d of the search is reached. A starting solution is selected as the center solution. A distance Δp is defined for each solution (permutation p of the center solution). The distance Δp is the number of facilities in p that are not in their center solution site. The Tabu list consists of solutions that are not further from the center solution, forcing the search away from the center solution.

If the iteration improves the best-found solution, the new best-found solution becomes the center solution and the next iteration begins. If the iteration fails to improve the best found solution we diversify up to L levels. The procedure terminates when L diversifications fail to improve the best-found solution. The post-merging procedure is presented in Figure 3.

- 1 Start with a center solution. It is also the best-found solution.
- 2 Set the level counter $c=0$.
- 3 Select the depth of the search d randomly in $[0.3n, 0.9n]$.

- 4 Set $\Delta p = 0$. The center solution is the starting solution sol_0 and the best-found solution. sol_1 and sol_2 (the best found solutions for $\Delta p + 1$ and $\Delta p + 2$, respectively) are emptied.
- 5 All pair exchanges of sol_0 are evaluated.
- 6 If the exchanged solution is better than the best-found solution, the best-found solution is updated and the rest of the exchanges are evaluated.
- 7 If the distance of the exchanged solution is Δp or less, the exchanged solution is placed in the Tabu list. It is thus ignored and the rest of the exchanges are evaluated.
- 8 If its distance is $\Delta p + 1$ or $\Delta p + 2$, sol_1 or sol_2 are updated, if necessary.
- 9 If a new best-found solution is found by scanning all the exchanges of sol_0 , the starting (center) solution is set to the new best-found solution. Go to Step 2.
- 10 Otherwise, set $sol_0 = sol_1$, $sol_1 = sol_2$, and sol_2 is emptied. Set $\Delta p = \Delta p + 1$.
- 11 If $\Delta p \leq d$, go to Step 5.
 - Otherwise
 - 11a Advance the level counter $c = c + 1$.
 - 11b If $c = L + 1$ stop and report the best-found solution. Otherwise,
 - 11c If c is odd use the best solution with depth d as the new center solution and go to Step 3.
 - 11d If c is even use the best solution found throughout the scan (the previous center solution is not considered) as the new center solution and go to Step 3.

Figure.3: Improvement procedure applied after cross-over operator.

3.1.3 The Compounded Approach

We employ the compounded genetic algorithm proposed in Drezner 2002c. We run the short search with $L=0$ ten times with a population size of $P=100$ and $\max\{2000, 40n\}$ generations. We collect the best ten population members at the termination of each run and create a starting population of $P=100$ for the short search with $L=6$ and $\max\{1000, 20n\}$ generations.

The computational complexity of one iteration is $O(n^3)$. The complexity of each algorithm depends on the expected number of iterations and the number of generations of the specific procedure.

3.2 Fast ant system (FANT)

The concept behind the ant system is to imitate the behavior of ants searching for food. This meta-heuristic was proposed by Coloni, Dorigo and Maniezzo (1992). Ants find 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 leave 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, adding 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 to the length of the path. Indeed, short paths will be

traveled at a higher rate than long ones and the amount of pheromone will grow faster on the short paths. Therefore, the ants are able to optimize their paths by this process.

A similar process can be applied to combinatorial optimization: solutions of the problem are built using a statistic on solutions previously built. This statistic plays the role of the pheromone traces and it gives higher weights to better solutions. After a while, it is observed that the constructive procedure is able to build solutions of better quality than a constructive procedure guided only by partial objective function evaluations.

Let us see how an ant system can be practically implemented for the QAP. Our FANT method exhibits a number of differences from the basic ant systems as described by Dorigo, Maniezzo and Colomi (1992). First, we have added a local search for improving the solution built by the constructive procedure. This has been identified as very promising for other problems such as the traveling salesman problem (Dorigo and Gambardella, 1997). Another difference is the statistic that is computed. Indeed, this statistic gives more and more weight to the best solution found so far as the search progresses. The search converges faster to good solutions. However, this may also cause the search to be trapped in local optima. In order to avoid this situation, a *diversification* mechanism that resets the memory has been developed. Finally, FANT does not use a population of artificial agents. We present the FANT method along the lines of adaptive memory procedures (Taillard et al., 1998, Taillard et al., 2001): Creation of a new solution using a memory (artificial pheromone), improvement of the new solution with a local search and memory update.

The basic idea of this method is to evaluate a priori the interest of setting $\pi_i = j$ in a solution (= permutation) π . Therefore, a matrix T of size $n \times n$, whose entry τ_{ij} measures the preference of setting $\pi_i = j$ is created. Initially, all entries of T are set to 1.

From an ant system point of view, this matrix represents the pheromone trail left by the ants. The entries of this matrix consist of a statistic derived from the solution previously generated by the search process. For generating a new (provisory) solution μ , 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 is used. More formally, the constructive method is presented in Figure 4:

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

Figure.4: Construction of a provisory solution.

The provisory solution μ generated at the first step is generally not so good because, in the first iteration, μ is just a random permutation. Therefore, the first steps of a first improving neighbor procedure

are applied to μ . 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. The improved solution is called π .

The statistic stored in memory is based on two parameters, r and r^* , that represent the reinforcement of the matrix entries corresponding to the solution π and, respectively, π^* , the best solution produced by the algorithm. During the entire process, r^* is a constant parameter set by the user while r may vary. Initially $r = 1$ and $\tau_{ij} = r$, $1 \leq i, j \leq n$, meaning that the memory does not initially contain any information. 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^* have been improved, then r is re-set to 1 and all the entries of T are set to 1. The aim of this re-setting 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 incremented 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 strategy is to diversify the search when the information contained in T is not much different from π^* .

In addition to the number of iterations to perform (= number of solutions built by artificial ants), the method has only one parameter, r^* .

4. Exact methods.

While great progress has been made on generating good solutions to large and difficult QAP instances, this has not been the case for finding exact solutions. The history of solving QAP instances exactly centers on the now famous Nugent problems. In 1968, Nugent, Vollmann and Ruml posed a set of problem instances of size 5, 6, 7, 8, 12, 15, 20 and 30, noted for their difficulty. In these instances, the distance matrix stems from an $n_1 \times n_2$ grid with Manhattan distance between grid points. Most of the resulting QAP instances have multiple global optima (at least four if $n_1 \neq n_2$ and at least eight if $n_1 = n_2$). Even worse, these globally optimal solutions are at the maximally possible distance from other globally optimal solutions. At the time of their paper, it was an achievement to find the optimum solution to the size 8 Nugent instance. Enumerating all 8! (eight factorial) possible assignments took 3374 seconds on a GE 265 computer. It was not until Burkard solved the Nug8 in 0.426 seconds on a CDC-CYBER76 machine (Burkard, 1975) that

notable progress was made toward exact solution methods. For this, Burkard used Gilmore-Lawler (GL) lower bounds (Gilmore, 1962) in a branch and bound algorithm.

In the 1970's and 80's, one could only expect to solve difficult instances for $n < 16$. In 1978 Burkard and Stratmann optimally solved the Nug12 in 29.325 seconds on a CDC Cyber76 machine. They used a branch-and-bound perturbation algorithm; again utilizing the popular GL lower bound. The GL was to remain the preferred lower bounding technique until well into the 1990's. In 1980, Burkard and Derigs reported that they were the first to solve the Nug15 using a branch-and-bound code. They did this in 2947.32 seconds on a CDC Cyber76.

It was not until 1997 that Clausen and Perregaard were able to solve exactly the very difficult Nug20 instance. For this, they used a parallel branch-and-bound algorithm on a MEIKO computing surface with 16 Intel i860 processors. The wall time for solving the 20 was 57,963 seconds and required the evaluation of 360,148,026 nodes. On a single processor the runtime would have been 811,440 seconds. Again, they relied upon the GL bound that had been developed decades earlier. Much progress has been made since then. Still, the exact solving of the most difficult of the Nugent instances, the Nug30 requires resources not commonly found at today's universities.

In the late 1990's two developments resulted in a large improvement in the ability to solve QAPs exactly. The first is a level-1 reformulation linearization technique (RLT) bound of Hahn and Grant (1998) that derives from a dual ascent procedure discovered much earlier (Hahn, 1968). Not only does the level-1 RLT yield strong bounds efficiently, but it also affords the removal of costs from branch sub-problems bringing them closer to dual solution and making the calculation of bounds within a branch-and-bound tree even more efficient (Hahn, Grant and Hall, 1998). Using this bound, Hahn, Hightower, Johnson, Guignard-Spielberg and Roucairol (2001) reported the general solution of the Nug25 in 1998. It took 5,698,818 seconds (66 days) on a single 360 MHz processor UltraSPARC 10 and required the evaluation of 108,738,131 nodes. The second development is the quadratic programming (QP) bound of Anstreicher and Brixius (2001a). It is the speed of calculation of the QP bound and its parallel implementation that makes it effective in solving the most difficult problems to date. These two methods respectively made it possible to solve exactly heretofore-unsolved problems of size 30, i.e., the Kra30a (Hahn and Krarup, 2001) and the Nug30 (Anstreicher, Brixius, Goux and Linderoth, 2001)

Hahn, with ideas from Hightower and Johnson, solved the Krarup30a in April 1999. The enumeration was done mainly on a Sun UltraSPARC10 with a single CPU. The total run time (adjusted to the Sun UltraSPARC10 speed) was 8,517,504 seconds (98.6 days). The number of nodes examined was 29,764,589.

Anstreicher, Brixius, Goux and Linderoth solved the Nug30 in June 2000 on a dynamic grid of workstations, massively parallel processors, and other CPUs using the Condor high-throughput computing system developed at the University of Wisconsin, together with tools from the Globus toolkit. The total wall-clock time was approximately 6.9 days. The number of worker machines averaged approximately 650, and peaked at 1009. Machines from eight universities participated in the computation. The worker machines

expended a total of 346 million CPU seconds. The equivalent computation time on a single HP-C3000 workstation would be approximately 6.9 years. Figure 5 shows the progress made in solving the Nugent instances.

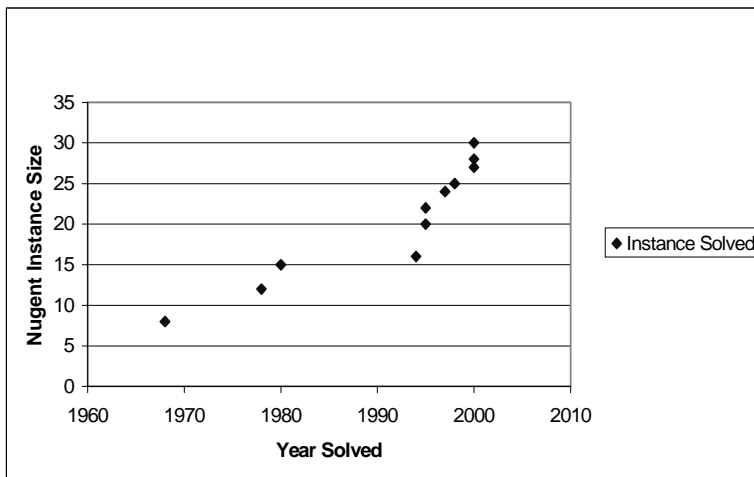


Figure 5: Nugent Instances First Solved

This past year, Hahn and Hightower (2002) developed a new lower bound for the Quadratic Assignment Problem (QAP) based on a level-2 reformulation-linearization technique (RLT). This technique provides even tighter bounds than the level-1 RLT formulation. Their bound is based on a formulation of the QAP similar to the one developed by Ramachandran and Pekny in 1996, which is based on the lifting of the problem into a higher dimensional space. However, the new technique has its primary roots in the level-2 RLT concept of Sherali and Adams (1990 and 1994). The method of calculation of this new level-2 RLT bound is a generalization of the dual ascent procedure developed for the level-1 RLT bound calculation (Hahn and Grant, 1998).

For large ($n > 20$) QAP instances that are difficult for exact methods, such as the Nugent instances, the level-2 RLT bound is a much better choice than either the Hahn-Grant level-1 RLT bound or the QP bound of Anstreicher and Brixius (see Hahn, Hightower, Johnson, Guignard-Spielberg and Roucairol 2002). The reason is that this bound is so much tighter. It takes longer to compute, but this doesn't matter, since far fewer nodes need to be evaluated.

Table 1 summarizes the performance of the three most successful exact solution algorithms for QAP instances size 27 to 36. These are the level-1 RLT of Hahn and Grant, the QP of Anstreicher and Brixius and the level-2 of Hightower and Hahn. The times are normalized to the speed of a single HPC3000 CPU. For the Nug27, the QP outperformed the Level-1 RLT by about 4.8-to-1, whereas for the Kra30b the Level-1 RLT outperformed the QP by about 2.6-to-1. For the Nug27, the Level-2 RLT outperformed the QP by 2.5-to-1 and for the Nug28 it outperformed the QP by 3.3-to-1.

Instance	Measurement	Level-1 RLT	QP	Level-2 RLT
Nug 27	No. of Nodes	297,648,966	~402,000,000	46,315
	Norm minutes	458,923	94,608	37,639
Nug 28	No. of Nodes	N/A	~2,230,000,000	~120,000
	Norm minutes	N/A	462,528	~140,000
Kra 30a	No. of Nodes	29,764,589	N/A	17,193
	Norm minutes	99,371	N/A	38,652
Kra 30b	No. of Nodes	183,659,980	5,136,036,412	N/A
	Norm minutes	367,200	1,403,352	N/A
Nug 30	No. of Nodes	N/A	11,892,208,412	N/A
	Norm minutes	N/A	3,647,664	N/A
Ste 36a	No. of Nodes	5,225,559	~1,790,000,000	N/A
	Norm minutes	27,649	36,540	N/A

N/A = Not available

Table 1: Comparison of Competing Branch-and-Bound Algorithms (times normalized to an HP C3000 cpu)

5. Computational experiments.

5.1 Computational experience with the hybrid genetic

The program using the compounded genetic algorithm was coded in MICROSOFT PowerStation 4.0 FORTRAN and run on a 600MHz Toshiba Portege 7200 laptop computer. Each instance was run 20 times. We report the minimum obtained, the number of times out of 20 that the minimum (or optimum) is obtained, the average percentage over the best-known (or optimal) solution, and the run time in minutes per run. Parameter settings are given in Section 3.1.3.

5.1.1 Results for Drex

Table 2 provides a few computational results concerning the Drex instances: Best and average solution values (absolute and relative to the optimum) obtained by the hybrid genetic algorithm. In this table, we see that the difficulty of the instances rapidly grows with their size. Also, we see that the experimental time complexity of our method grows as $O(n^4)$.

Problem	Opt.	Minimum			Average		
		Value	%/Opt.	# Times	Value	%/Opt.	Time
Dre30	508	508	0	20	508.0	0	2.39
Dre42	764	764	0	18	774.2	1.34	9.13
Dre56	1086	1086	0	6	1275.2	17.46	30.18
Dre72	1452	1452	0	2	1848.1	27.28	93.19
Dre90	1838	2218	20.67	0	2460.7	33.88	192.63

Table 2: Results for hybrid genetic on Drex instances

5.1.2 Results for Taixxey

The best-known solution for all Tai27eyy and Tai45eyy problems was found in all 20 runs. The run times are about 1 and 5 minutes per run, respectively, on the average with very small variation. The results

for Tai75eyy are summarized in Table 3. Run times are about 37 minutes per run. In this table, we see that the genetic hybrid perform relatively well on these instances, missing the best known solution in only 5% on the average. However, let us mention that the computational times are relatively high (it took about 10 CPU days to gather the results presented in Table 3). Therefore, it would be interesting to evaluate the performances of the method while running fewer generations. A general method for comparing non-deterministic iterative searches is presented in the next section. This method follows the lines of what is proposed in Taillard (2001 and 2002).

Problem	Best Known	Times BK found	Average	% over BK
Tai75e01	14488	14	14513.4	0.175
Tai75e02	14444	14	14493.0	0.339
Tai75e03	14154	20	14154.0	0
Tai75e04	13694	19	13698.1	0.03
Tai75e05	12884	20	12884.0	0
Tai75e06	12534	20	12534.0	0
Tai75e07	13782	17	13800.7	0.136
Tai75e08	13948	19	13956.2	0.059
Tai75e09	12650	20	12650.0	0
Tai75e10	14192	19	14213.1	0.149
Tai75e11	15250	20	15250.0	0
Tai75e12	12760	20	12760.0	0
Tai75e13	13024	20	13024.0	0
Tai75e14	12604	20	12064.0	0
Tai75e15	14294	20	14294.0	0
Tai75e16	14204	19	14211.8	0.055
Tai75e17	13210	17	13227.1	0.129
Tai75e18	13500	20	13500.0	0
Tai75e19	12060	20	12060.0	0
Tai75e20	15260	19	15268.1	0.053
Average:		18.85		0.056

Table 3: Results for hybrid genetic on Tai75eyy instances.

5.2 Comparing non-deterministic iterative methods

Comparing two (or more) heuristic methods based on metaheuristic principles is a difficult task that is not solved yet in a satisfactory way. Indeed, these methods are iterative, meaning that the longer they run, the better the solutions they produce. They are also almost all non-deterministic, since they use a pseudo-random number generator. This means that it is possible to obtain two different solutions when running the same heuristic method twice.

It is clear that comparing iterative methods must be done considering both the quality of the solution produced and the computational effort. The last is traditionally the running time on a given machine. Unfortunately, this measure is both imprecise and volatile. Indeed, the running time depends on the

programming style, on the compiler, on the compiling options, etc. Moreover, the lifetime of computer equipment is very limited. Sometimes, the computer is already obsolete when the articles describing a new method are published!

If the combinatorial problem consists of optimizing a unique objective function, it is easy to compare the quality of two solutions. However, the comparison of non-deterministic methods is not so easy. The solution quality not only depends on the computational effort but also on the pseudo-random number generator seed. Suppose that heuristic method **A** is compared to heuristic method **B** and **A** and **B** were run n_A and n_B times, respectively. During these runs, each improvement of solution quality is recorded with the corresponding computational effort such that, for a given time t , the quality of the $n_A + n_B$ solutions produced by both methods is known. We wish to determine whether **A** is better than **B** at time t .

The solution quality obtained by methods **A** and **B**, respectively, at time t are random variables $X_{A(t)}$ and $X_{B(t)}$, respectively. The probability density functions of these random variables are $f_{A(t)}$ and $f_{B(t)}$, respectively. We wish to compare the mathematical expectations $E(X_{A(t)})$ and $E(X_{B(t)})$ in order to know which one is lower and decide which method is better for time t . Unfortunately, $f_{A(t)}$ and $f_{B(t)}$ are unknown. Moreover, the number of runs n_A and n_B are typically limited to a few dozen. Therefore, it is not tractable to attempt the determination of the density functions $f_{A(t)}$ and $f_{B(t)}$ with the data available. We cannot assume (as is often done in literature) that these curves are normally distributed (for instance, these functions are bounded by the value of the global optimum while normal distributions are unbounded). Therefore, a comparison of both methods on the basis of average and standard deviation of solution values observed is in general not appropriate.

Therefore, other hypothesis tests have to be used. The Mann-Whitney test allows a proper comparison: translated to the comparison of a non-deterministic heuristic method, the null hypothesis is $f_{A(t)} = f_{B(t)}$. If the probability that this hypothesis is true is lower than a given confidence level α , considering the n_A and n_B solutions obtained, then the null hypothesis is rejected and the alternative hypothesis is accepted, i.e., the probability that **A** produces a better solution than **B** is larger (or lower) than the probability that **B** produces a better solution than **A**.

In short, the Mann-Whitney test is performed as follows: first, the $n_A + n_B$ solutions are ranked by decreasing quality and a rank between 1 and $n_A + n_B$ is attributed to each solution. If many solutions have the same value of the objective function, they all get the same rank, i.e., the average of the rank they would have obtained if there were no equality. Then, a statistic $S_{A(t)}$ is computed. This statistic corresponds to the sum of the ranks of solutions issued from method **A**. If $S_{A(t)} > T_{\alpha}(n_A, n_B)$, then the null hypothesis is rejected with a confidence level of α and it is accepted that **A** is worse than **B**. The values of $T_{\alpha}(n_A, n_B)$ can be found in tables; there are also books that provide tables that give α as a function of $S_{A(t)}$ and $n_A + n_B$. More details on the Mann-Whitney test can be found in Conover (1999).

This test has to be repeated for different values of computational time t . A convenient way to compare methods **A** and **B** is to graphically draw the probability that **A** is better than **B** as a function of the

computational effort. Naturally, such a comparison is possible only if reliable computational times can be obtained. However, it could happen that one method could be implemented in such a way that it runs faster. Therefore, we have to be particularly careful when comparing computing times, for instance, by considering a multiplicative safety factor in the measure of computing times.

Finally, as mentioned above, the computing times are an essential criterion when comparing heuristic methods, but this criterion is unreliable. If one wants to be independent of the computer used for presenting the comparisons, it is required to consider an absolute computational effort, such as the number of iterations. When one can derive the mathematical complexity for a single iteration, a more reliable computational measure can always be obtained.

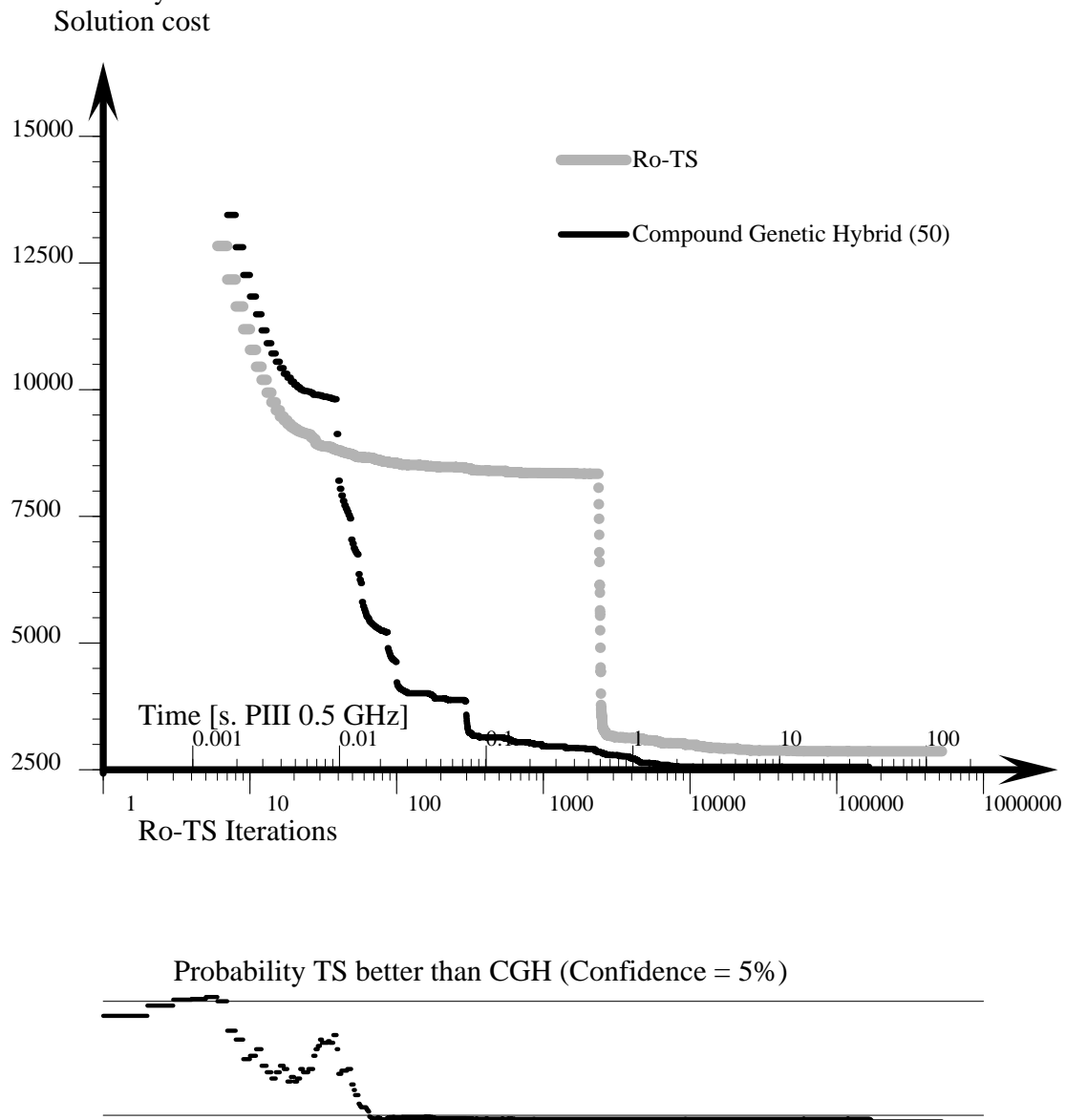


Figure 6: Comparison of Robust Tabu Search and Compounded Hybrid Genetic on instance Tai27e01.

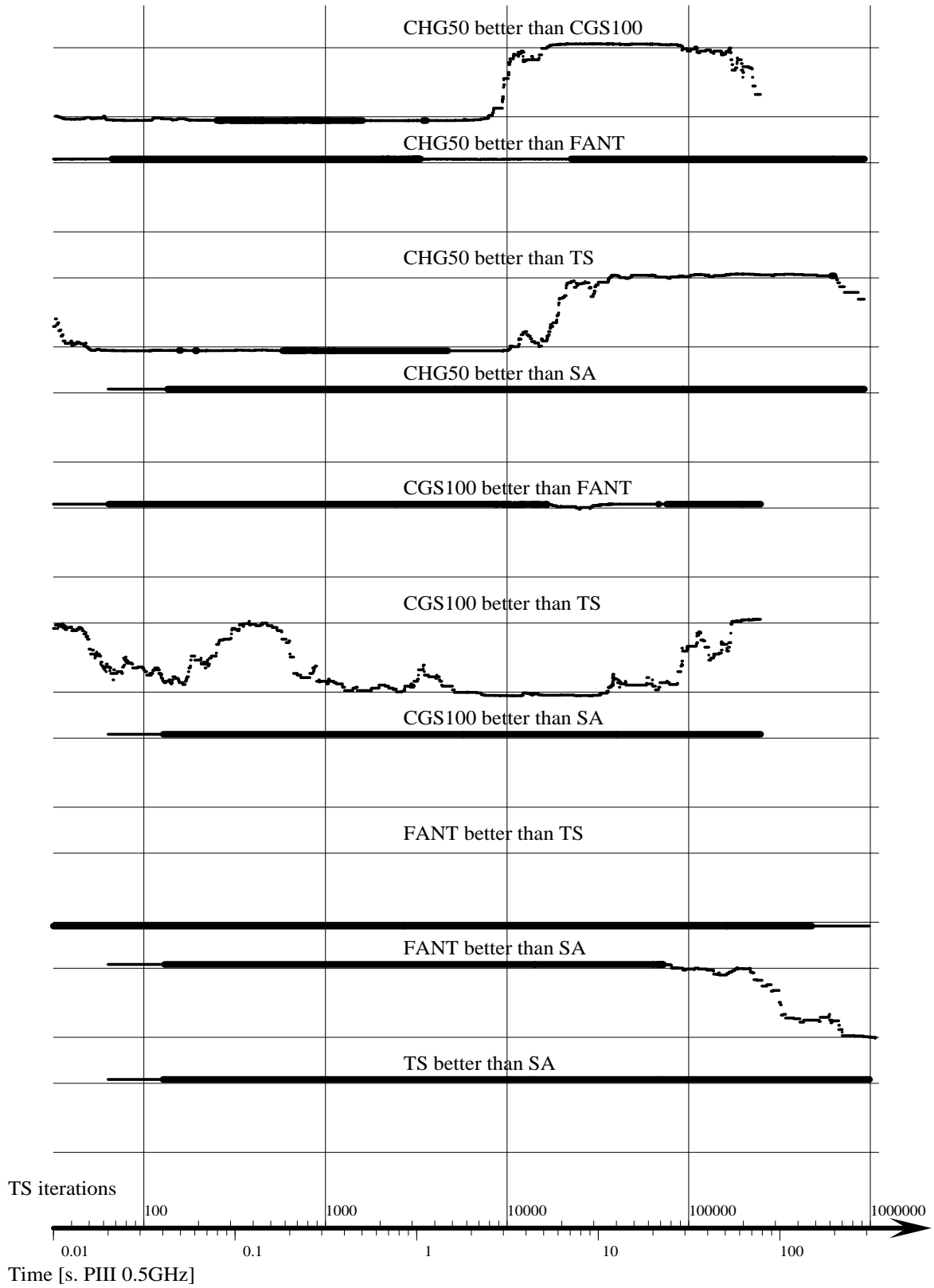


Figure 7: Probability diagrams (confidence: 5%-95%) comparing five different methods all together on instance Dre30

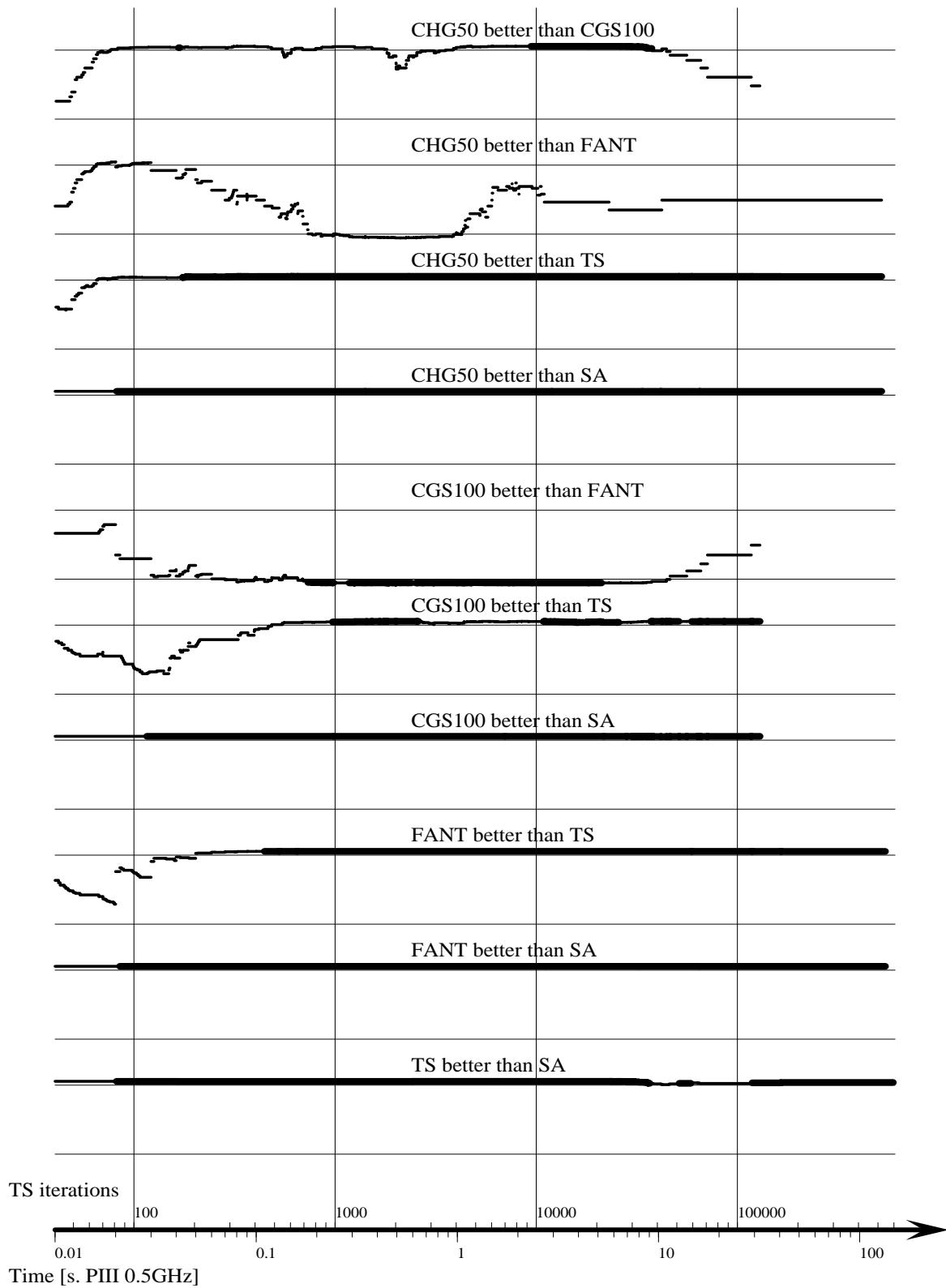


Figure 8: Probability diagrams (confidence: 5%-95%) comparing five different methods all together on instance Tai27e01

In Figure 6, these principles have been used to compare the robust tabu search of Taillard (1991) with the compounded hybrid genetic method (with $P=50$) on our new problem instance Tai27e01. We see in this figure that the solution values obtained at the beginning of the search are very bad for both methods. Ro-TS is trapped for many iterations in bad local optima while the compounded genetic hybrid finds the structure of good solutions earlier. A second diagram is depicted in this figure plotting the probability of Ro-TS to be better than the hybrid genetic method. The horizontal lines in this diagram indicate the probabilities 0.05 and 0.95. We observe in the second diagram that the genetic hybrid method is significantly better than Ro-TS (at a confidence level of 5%) as soon as the computational effort reaches the equivalent of about 100 Ro-TS iterations. However, for small computational efforts, Ro-TS can be significantly better than the hybrid genetic.

The variance of the solution value (not shown in Figure 6 for clarity reasons) is very high (except for large computational efforts, when almost all runs have found the optimum). Therefore, a classical (parametric) statistical test for comparing the average of both methods cannot show the superiority of a method, while the non-parametric test we used is often able to show it. Therefore, we note that the second (probability) diagram contains a lot of information, and the most basic and interesting question: Is method *A* significantly better than method *B* after computational effort t ?

For comparing two methods, this second diagram is much more valuable than a classical table reporting average (and best, worst, standard deviation, etc.) of the solution values for a given computational effort, and is more compact. By putting several of such diagrams in the same figure, it is possible to compare many heuristic methods in one plot. This was done in Figures 7 and Figure 8, respectively, in which we compare problem instances Dre30 and Tai27e01, respectively: the simulated annealing of Connolly (1990), the tabu search of Taillard (1991), FANT with parameter $r^*=5$, compounded hybrid genetic with parameter $P=50$ and genetic short search with $P=100$. From these two figures, we conclude that SA is significantly worse than the other methods (for these two problem instances) while none of the others is clearly better for all values of t .

5.3 Exact methods

While the new Drex and Taixe instances were indeed difficult for heuristic solution methods, it was not surprising to find them easy for exact methods. It was, however, surprising that these instances solved most easily using the level-1 RLT lower bound method of Hahn and Grant. One would think that the level-2 RLT method would be better, since it produces tighter lower bounds.

Both the level-1 and level-2 RLT approaches were tried. But, it was determined early in the experimentation that the level-1 RLT gave sufficiently tight bounds at the root and was efficient for branch-and-bound enumeration. On the other hand, the level-2 RLT bounds, which were indeed tighter, took longer to calculate; so that the resulting enumeration runtimes were much longer. For instance, the Tai27e03 enumeration took 667.2 seconds using the level-2 RLT; whereas using the level-1 RLT on this instance took

only 57.6 seconds. Thus, the results reported here are for only the level-1 RLT calculations. The results of the experiments using exact methods are presented in Tables 4 through 7 below.

Instance	Optimum	100 Iterations		500 Iterations	
		Normalized Time (secs)	Bound	Normalized Time (secs)	Bound
Dre18	332	4.5	331.1	22.3	331.5
Dre21	356	13.0	354.2	64.1	355.5
Dre24	396	19.0	393.4	89.7	394.9
Dre28	476	59.7	473.2	295.0	474.9
Dre30	508	72.2	503.7	356.5	506.1
Dre42	764	401.3	753.3	1,955.1	760.0
Dre56	1,086	386.9	1,055.1	1,885.3	1,069.7
Dre72	1,452	2,707.8	1,398.9	16,373.0	1,425.0

Table 4: Level-1 RLT Lower Bounds for Drexx Instances (time normalized to HP C3000 cpu - machines in Table 5)

Table 4 lists the lower bound calculations on the original problems for a selected set of the Drexx instances. Table 5 lists the exact solution (branch-and-bound enumeration) runtimes and number of nodes evaluated for the same set of Drexx instances, as in Table 4. Table 6 lists the lower bound calculations and the exact solution runtimes and number of nodes evaluated for all 20 of the Tai27exx instances. All runs reported in Table 6 were made on a Sun Ultra 10 with single a 366 MHz processor. Table 7 lists the lower bound calculations and the exact solution runtimes and number of nodes evaluated for the Tai45e01 and Tai75e01 instances. In this table, the runtimes are normalized to the speed of a single HP C3000 processor.

Instance	No. of Nodes	Normalized Time(secs)	Memory (Mbytes)	Machine	Speed Factor
Dre18	144	1.1	13	Sun Ultra 10	0.68
Dre21	409	3.9	20	Sun Ultra 10	0.68
Dre24	578	7.8	33	Sun Ultra 10	0.68
Dre28	909	31.7	69	Sun Ultra 10	0.68
Dre30	929	49.7	93	Sun Ultra 10	0.68
Dre42	2,165	496.9	417	HPJ5000	1.40
Dre56	8,299	7,411.9	1,041	SGIO 2000 (UI)	0.55
Dre72	33,277	215,510.8	3,250	SGIO 2000 (UP)	0.63

Table 5: Level-1 B & B Enumeration for Drexx Instances (time normalized to HP C3000 cpu - see speed factor)

Tai27 Instance Optimum		Lower Bound at Root				Level 1 B&B Enumeration**		
		100 Iterations		500 or less iterations		No. of Nodes	runtime (secs)*	No. of optima
		runtime (secs)*	Bound	runtime (secs)*	Bound			
e01	2,558	26.0	2,554	Solved at root		1	40.1	n/a
e02	2,850	22.3	2,795	104.3	2,836	240	22.5	2
e03	3,258	23.8	3,257	46.2	3,258	207	57.6	2
e04	2,822	23.0	2,737	109.7	2,795	103	24.9	1
e05	3,074	23.5	3,044	45.8	3,074	553	94.1	4
e06	2,814	23.2	2,814	n/a	n/a	1	47.1	n/a
e07	3,428	23.3	3,354	110.2	3,422	78	13.4	1
e08	2,430	22.6	2,429	43.6	2,429	242	50.7	2
e09	2,902	22.3	2,796	104.9	2,838	53	25.5	1
e10	2,994	22.5	2,994	Solved at root		1	25.9	n/a
e11	2,906	22.3	2,895	Solved at root		1	28.8	n/a
e12	3,070	22.6	3,070	Solved at root		1	71.9	n/a
e13	2,966	22.3	2,825	105.0	2,872	333	63.2	2
e14	3,568	22.5	3,491	105.3	3,539	168	20.3	1
e15	2,628	22.2	2,603	104.5	2,608	323	51.9	2
e16	3,124	Solved at root		Solved at root		1	20.0	n/a
e17	3,840	22.3	3,514	105.2	3,600	718	115.2	1
e18	2,758	22.3	2,752	104.0	2,755	78	11.0	1
e19	2,514	22.3	2,495	Solved at root		1	49.9	n/a
e20	2,638	22.4	2,623	Solved at root		1	65.1	n/a

*Runtime on a Sun Ultra 10 (multiply runtime by 0.68 to get time on HP C3000.

**Requires 45 MB of RAM.

n/a = not available.

Table 6: Lower Bounds and B&B Runtimes for Tai27exx Instances

Instance	Optimum	Lower Bound at Root				B & B Enumeration		
		100 Iterations		500 Iterations		No. of Nodes	Time (secs)	Memory (Mbytes)
		Time (secs)	Bound	Time (secs)	Bound			
Tai45e01	6,412	205.5	6,383	solved at root		1	361.5	498
Tai75e01	14,488	1,375.2	13,654	6,776	14,096	58,631	348,186	3,950

Table 7: Level-1 RLT Runtimes for Large Taillard Instances (times normalized to HP C3000 cpu)

6. Discussion

Two types of quadratic assignment problems were generated that are difficult for common heuristic techniques, but that turned out to be relatively easy for a modern exact algorithm. We

tested these problems for various metaheuristic algorithms. The exact algorithm solved many of these problems (with up to 75 facilities) in reasonable computer time.

We also proposed a statistical analysis to compare non-deterministic heuristic methods and prove statistically that one method is better than the other. This technique is universal and may prove very useful for comparisons of heuristic techniques for the solution of other problems as well.

Acknowledgements

Part of this research was conducted while the first author was visiting the Graduate School of Management, University of California at Irvine.

The work by the second author was supported in part by an international travel grant INT-9900376 from the National Science Foundation.

7. Bibliography

K.M. Anstreicher and N.W. Brixius, “A New Bound for the Quadratic Assignment Problem Based on Convex Quadratic Programming,” *Mathematical Programming*, vol. 89, 341-357, 2001a.

K.M. Anstreicher and N.W. Brixius, “Solving quadratic assignment problems using convex quadratic programming relaxations,” *Optimization Methods and Software*, vol. 16, 49-68, 2001b.

K.M. Anstreicher, N.W. Brixius, J.-P. Goux and J. Linderoth, “Solving large quadratic assignment problems on computational grids,” to appear in *Mathematical Programming*, Series B, 2001, Currently available on the Web from <http://www.biz.uiowa.edu/faculty/anstreicher/mwqap.ps>

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

R.E. Burkard, “Numerische Erfahrungen Mit Summen - Und Bottleneck-Zuordnungsproblemen,” in *Numerische Methoden Bei graphentheoretischen und kombinatorischen Problemen*, Eds. L. Collatz and H. Werner, Birkhauser Verlag, Basel, 1975

R. E. Burkard, and U. Derigs, “Assignment and Matching Problems,” *Solution Methods with FORTRAN Programs*, New York, Springer-Verlag, 1980.

R. E. Burkard, E. Çela, S.E. Karisch and F. Rendl, QAPLIB — A quadratic assignment problem library, *European Journal of Operational Research* 55, 1991, 115-119, electronic update: <http://www.opt.math.tu-graz.ac.at/qaplib>, (29. 1. 1997).

R. E. Burkard, and F. Rendl, A thermodynamically motivated simulation procedure for combinatorial optimization problems, *European Journal of Operational Research* 17, 1984, 169-174.

- R.E. Burkard, and K.H Stratmann**, “Numerical investigations on quadratic assignment problems,” *Naval Research Logistical Quarterly*, vol. 25, no. 1: 129-148, 1978.
- J. Clausen, and M. Perregaard**, “Solving large quadratic assignment problems in parallel,” *Computational Optimization and Applications*, 8: 111-128 1997.
- A. Colorni, M. Dorigo and V. Maniezzo**, Distributed Optimization by Ant Colonies, *Proceedings of ECAL'91 — European Conference on Artificial Life*, Elsevier Publishing, 1992, 134-142.
- D. T. Connolly**, “An improved annealing scheme for the QAP”, *European Journal of Operational Research* 46, 1990, 93-100.
- W. J. Conover**, *Practical Nonparametric Statistics*, 3. ed. Wiley, 1999.
- 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.
- M. Dorigo and L. M. Gambardella**, Ant colony system: A cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Trans. Evolutionary Computing* 1, 1997.
- Z. Drezner**, “Heuristic Algorithms for the Solution of the Quadratic Assignment Problem,” *Journal of Applied Mathematics and Decision Sciences*, 6, 163–173, 2002a.
- Z. Drezner**, “A New Genetic Algorithm for the Quadratic Assignment Problem,” *INFORMS Journal on Computing*, in press, 2002b.
- Z. Drezner**, “Robust Heuristic Algorithms for the Quadratic Assignment Problem,” under review, 2002c.
- A.N. Elshafei**, “Hospital lay-out as a quadratic assignment problem”, *Operational Research Quarterly* 28, 1977, 167–179.
- C. Fleurent and J. Ferland**, “Genetic hybrids for the quadratic assignment problem”, *DIMACS Series in Math. Theoretical Computer Science* 16, 1994, 190-206.
- L. M. Gambardella, E. D. Taillard and M. Dorigo**, “Ant colonies for the quadratic assignment problem”, *Journal of the Operational Research Society* 50, 1999, 167-176. .
- P.C. Gilmore**, “Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem,” *Journal of the Society of Industrial and Applied Mathematics*, vol. 10, no. 2, pp. 305-313, 1962.
- P.M. Hahn**, “Minimization of Cost in Assignment of Codes to Data Transmission,” *Ph.D. Dissertation*, University of Pennsylvania, 1968.
- P.M. Hahn, and T.L. Grant**, “Lower Bounds for the Quadratic Assignment Problem Based Upon a Dual Formulation,” *Operations Research*, vol. 46, no. 6, Nov-Dec, pp. 912-922, 1998.

P.M. Hahn, T.L. Grant and N. Hall, “A Branch-and-Bound Algorithm for the Quadratic Assignment Problem Based on the Hungarian Method,” *European Journal of Operational Research*, vol. 108, pp. 629-640, 1998.

P.M. Hahn, W.L. Hightower, T.A. Johnson, M. Guignard-Spielberg and C. Roucairol, “Tree elaboration strategies in branch-and-bound algorithms for solving the quadratic assignment problem,” *Yugoslav Journal of Operations Research*, vol. 11 No.1, pp. 41-60, 2001.

P.M. Hahn, W.L. Hightower, T.A. Johnson, M. Guignard-Spielberg and C. Roucairol, “A Lower Bound for the Quadratic Assignment Problem Based on a Level-2 Reformulation-Linearization Technique,” Submitter for publication in *Computers in Operations Research*, 2002.

P.M. Hahn and J. Krarup, “A hospital facility problem finally solved,” *The Journal of Intelligent Manufacturing*, vol. 12, No. 5/6, pp 487-496, 2001. Also on web site: <http://www.seas.upenn.edu/~hahn/>

P. Hansen and N. Mladenovic, “Variable neighborhood search: Principles and applications,” *European Journal of Operational Research*, 130, 2001, 449–467.

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

G. Laporte and H. Mercure, “Balancing hydraulic turbine runners: A quadratic assignment problem”, *European Journal of Operational Research* 35, 1988, 378-381.

N. Mladenovic and P. Hansen, “Variable Neighborhood Search,” *Computers and Operations Research*, 24, 1997, 1097-1100.

P. Moscato, “Memetic Algorithms,” in P.M. Pardalos and M.G.C. Resende (Eds.) *Handbook of Applied Optimization*, Oxford University Press, Oxford, U.K. 2002.

C.E. Nugent, T.E. Vollman and J. Ruml, “An Experimental Comparison of Techniques for the Assignment of Facilities to Locations,” *Operations Research*, vol. 16, pp. 150-173, 1968.

R. Ramachandran and J.F. Pekny, “Dynamic Factorization Methods for Using Formulations Derived from Higher Order Lifting Techniques in the Solution of the Quadratic Assignment Problem,” *State of the Art in Global Optimization: Computational Methods and Applications*, Kluwer Academic Publishers, pp. 75-92, 1996

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

H.D. Sherali and W.P. Adams, “A Hierarchy of Relaxations between the Continuous and Convex Hull Representations for Zero-One Programming Problems,” *SIAM Journal on Discrete Mathematics*, vol. 3, No. 3, pp. 411-430, 1990.

- H.D. Sherali and W.P. Adams**, “A Hierarchy of Relaxations and Convex Hull Characterizations for Mixed-Integer Zero-One Programming Problems,” *Discrete Applied Mathematics*, vol. 52, no. 1, pp. 83-106, 1994.
- J. Skorin-Kapov**, Tabu search applied to the quadratic assignment problem, *ORSA Journal on Computing* 2, 1990, 33-45.
- 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.
- L. Steinberg**, “The backboard wiring problem: A placement algorithm”, *SIAM Review* 3, 1961, 37–50.
- T. Stützle and H. H. Hoos**, “The MAX-MIN Ant System and Local Search for Combinatorial Optimization Problems: Towards Adaptive Tools for Global Optimization”, in: *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I.H. Osman, C. Roucairol (editors), Kluwer Academic Publishers, 1999, 313-329.
- É. D. Taillard**, “Robust tabu search for the quadratic assignment problem”, *Parallel Computing* 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**, "Comparison of Non-Deterministic Iterative Methods", *Proceedings of MIC'2001 - 4th Metaheuristic International Conference*, 2001, 272-276.
- É. D. Taillard**, "Principes d'implémentation des métaheuristiques", in: M. Pirlot and J. Teghem, *Métaheuristiques et outils nouveaux en recherche opérationnelle: Méthodes*, Hermès, 2002, 55-77.
- É. D. Taillard, L. M. Gambardella, M. Gendreau and J.-Y. Potvin** “Programmation à mémoire adaptative”, *Calculateurs Parallèles, Réseaux et Systèmes répartis* 10, 1998, 117-140
- É. D. Taillard, L. M. Gambardella, M. Gendreau and J.-Y. Potvin** "Adaptive Memory Programming: A Unified View of Meta-Heuristics", *European Journal of Operational Research* 135 (1), 2001, 1-16.
- D. E. Tate and A. E. Smith**, “A genetic approach to the quadratic assignment problem”, *Computers and Operations Research* 1, 1995, 855-865.