A GRASP for the Biquadratic Assignment Problem*

T. Mavridou† P.M. Pardalos† L. Pitsoulis†
M.G.C. Resende‡

Abstract

The biquadratic assignment problem (BiQAP) is a generalization of the quadratic assignment problem (QAP). It is a nonlinear integer programming problem where the objective function is a fourth degree multivariable polynomial and the feasible domain is the assignment polytope. BiQAP problems appear in VLSI synthesis. Due to the difficulty of this problem, only heuristic solution approaches have been proposed. In this paper, we propose a new heuristic for the BiQAP, a greedy randomized adaptive search procedure (GRASP). Computational results on instances described in the literature indicate that this procedure consistently finds better solutions than previously described algorithms.

Keywords: Heuristics, combinatorial optimization, nonlinear assignment problem, local search, GRASP

1 Introduction

The biquadratic assignment problem (BiQAP) is a generalization of the quadratic assignment problem (QAP). It is a nonlinear integer programming problem where the objective function is a fourth degree multivariable polynomial and the feasible domain is the assignment polytope. Motivated by a practical problem arising in VLSI synthesis, the problem was first introduced and studied by Burkard, Çela and Klinz [2].

The BiQAP can be stated as follows:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n}\sum_{m=1}^{n}\sum_{p=1}^{n}\sum_{s=1}^{n}\sum_{t=1}^{n} a_{ijkl}b_{mpst}x_{im}x_{jp}x_{ks}x_{lt}$$

*May 1, 1996

†Center of Applied Optimization, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611 USA

‡Information Sciences Research Center, AT&T Research, Murray Hill, NJ 07974 USA

subject to:

$$\begin{aligned}
\sum_{i=1}^{n} x_{ij} &= 1, \quad j = 1, 2, \ldots, n, \\
\sum_{j=1}^{n} x_{ij} &= 1, \quad i = 1, 2, \ldots, n, \\
x_{ij} &\in \{0, 1\}, \quad i, j = 1, 2, \ldots, n,
\end{aligned}$$

where the fourth-dimensional arrays $A = (a_{ijkl})$ and $B = (b_{mpst})$ each have n^4 elements. Equivalently, the BiQAP can be stated as:

$$\min_{p \in \Pi_N} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} a_{ijkl} b_{p(i)p(j)p(k)p(l)},$$

where Π_N denotes the set of all permutations of $N = \{1, 2, \ldots, n\}$. The latter formulation of the BiQAP will be used throughout this paper.

Burkard, Çela and Klinz [2] showed that the BiQAP is NP-hard. They computed lower bounds for BiQAP derived from lower bounds of the QAP. The computational results showed that these bounds are weak and deteriorate as the dimension of the problem increases. This observation suggests that branch and bound methods will only be effective on very small instances. For larger instances, efficient heuristics, that find good-quality approximate solutions, are needed.

Burkard and Çela [1] developed several heuristics for the BiQAP, in particular deterministic improvement methods and variants of simulated annealing and tabu search. Computational experiments on test problems with known optimal solutions [2], suggest that one version of simulated annealing is best among those tested.

In this paper, we describe a Greedy Randomized Adaptive Search Procedure (GRASP) for the BiQAP and examine its performance on the set of BiQAP instances considered in [1]. A GRASP [3] is an iterative randomized metaheuristic that has been applied to many combinatorial optimization problems, including the QAP [6, 7, 8, 9].

The remainder of the paper is organized as follows. In Section 2, we describe the GRASP for BiQAP and discuss the relation of this heuristic to GRASPs for other nonlinear assignment problems. Experimental results are presented in Section 3. Concluding remarks are made in Section 4.

2 GRASP for the BiQAP

GRASP is an iterative procedure consisting of two phases at each iteration, a construction phase and a local search phase. In the first phase, the randomized

```
procedure GRASP(ListSize,MaxIter,RandomSeed)
1    InputInstance();
2    do k = 1,...,MaxIter →
3        ConstructGreedyRandomizedSolution(ListSize,RandomSeed);
4        LocalSearch(BestSolutionFound);
5        UpdateSolution(BestSolutionFound);
6    od;
7    return BestSolutionFound
end GRASP;
```

Figure 1: Pseudo-code for a generic GRASP

algorithm constructs a solution, guided by a greedy function. In the second phase, local search is applied in the neighborhood of the constructed solution, for possible further improvement. This process is repeated until a stopping criterion, such as maximum number of iterations, is satisfied. GRASP can be viewed as a procedure that samples (greedily-biased) high-quality points from the solution space, searching the neighborhood of each point for a local optimum. For a tutorial and survey of GRASP, see Feo and Resende [3]. Figure 1 shows pseudo-code for a generic GRASP.

GRASP has been implemented to solve QAPs with dense [6, 9] and sparse coefficient matrices [7]. A parallel implementation of a GRASP for QAP is described in [8].

The feasible space for a BiQAP of size n is the same as for a QAP of that size, i.e. the set of all possible $n!$ permutations of the set $N = \{1, 2, \ldots, n\}$. Each of the n components of the permutation vector corresponds to an assignment. During the GRASP construction phase a complete permutation is constructed. Initially, four assignments are made simultaneously. After that, one assignment is made at a time. Each new assignment added to a partial permutation contributes to the total cost of the assignment the cost of its interaction with already-made assignments. To construct the permutation, costs are computed for each feasible assignment and among those with small cost, one is randomly selected to be added to the partial permutation. As is the case in the GRASP for QAP, the construction phase of the GRASP for BiQAP has two stages. The first stage makes the four initial assignments simultaneously, while the second stage makes the remaining $n-4$, one assignment at a time. In the GRASP local search phase, 2-exchange local search is applied to the permutation constructed in the first phase.

In the remainder of this section, we provide a detailed description of the components of the GRASP for the BiQAP.

2.1 Stage 1 of the Construction Phase

Stage 1 of the GRASP construction phase makes the first four assignments simultaneously. For these assignments, the greedy choice is the set of four assignments with the minimum cost of interaction. Note that the minimum feasible number of initial assignments is four, since the BiQAP is only defined for problems with dimension $n \geq 4$ and costs can only be computed if at least four assignments are made.

To select the four assignments, the interaction cost of each set of four assignments must be computed. The sets of assignments with small costs are placed in a restricted candidate list (RCL) and one set is selected at random from the RCL. Consider the computation of the interaction cost of four assignments in a BiQAP instance with input coefficient arrays $A = (a_{ijkl})$ and $B = (b_{mpst})$ of n^4 elements each. More specifically, the interaction cost of the set $\mathcal{A}$ of four assignments is

$$C_{\mathcal{A}} = \sum_{i=1}^{4}\sum_{j=1}^{4}\sum_{k=1}^{4}\sum_{l=1}^{4} a_{p_{\mathcal{A}}(i)p_{\mathcal{A}}(j)p_{\mathcal{A}}(k)p_{\mathcal{A}}(l)} b_{q_{\mathcal{A}}(i)q_{\mathcal{A}}(j)q_{\mathcal{A}}(k)q_{\mathcal{A}}(l)}, \qquad (1)$$

where the injections $p_{\mathcal{A}}, q_{\mathcal{A}}$ are defined as the quadruples:

$$p_{\mathcal{A}}, q_{\mathcal{A}} \in K = \{(i,j,k,l) \mid i,j,k,l = 1,2,\ldots,n,\ \ i \neq j,k,l,\ j \neq k,l,\ \ k \neq l\}.$$

The number of feasible sets of four assignments is $|K|^2$, and it is easily seen that $|K| = n(n-1)(n-2)(n-3)$. The RCL contains all sets $\mathcal{A}$ of four assignments that have small interaction costs $C_{\mathcal{A}}$. This RCL for Stage 1 needs only to be setup once, since it does not change from one GRASP iteration to the next.

The following procedure is used to create the RCL. Let α, β $(0 \leq \alpha, \beta \leq 1)$ be the RCL parameters, $\lfloor x \rfloor$ denote the largest integer smaller or equal to x, and Λ $(1 \leq \Lambda \leq |K|^2)$ be the maximum number of generated cost parameters. Since there are $\mathcal{O}(n^8)$ sets of four assignments, generation of all sets is time consuming. Instead, the algorithm generates Λ random permutation pairs, and for each pair $p_{\mathcal{A}}$ and $q_{\mathcal{A}}$ it computes the associated interaction cost $C_{\mathcal{A}}$. The costs are sorted in increasing order, and the $\lfloor \beta\Lambda \rfloor$ smallest elements are considered as potential candidates. The RCL is made up of the $\lfloor \alpha\beta\Lambda \rfloor$ smallest elements. Note again that no change will be made in the order of these elements during the GRASP iterations, and therefore this phase of the GRASP can be performed in constant time, since it simply consists of selecting, at random, an element from a set. The procedure described above is implemented efficiently using heap sort.

2.2 Stage 2 of the Construction Phase

In Stage 2 of the GRASP construction phase, the final $n-4$ assignments are made, one assignment at a time. Define the set Γ_r as the set of already-made

assignments prior to making the r-th assignment ($r \geq 5$):

$$\Gamma_r = \{(j_1, p_1), (j_2, p_2), \ldots, (j_{r-1}, p_{r-1})\}.$$

At the start of Stage 2, $|\Gamma_5| = 4$, since four assignments are made in Stage 1 of the construction phase. Throughout the stage, $r = |\Gamma_r| + 1$.

The $U = (n - r + 1)^2$ possible assignments of m to s need to be examined to include in the RCL only those with small cost with respect to the assignments in set Γ_r. One assignment in the RCL will be selected at random. One trivial way to determine the cost of assigning m to s is to compute the sum

$$C_{ms} = \sum_{(i,j,k,l) \in T} a_{p(i)p(j)p(k)p(l)} b_{q(i)q(j)q(k)q(l)}, \tag{2}$$

where

$$T = \{(i, j, k, l) \mid i, j, k, l = 1, 2, \ldots, r\},$$

$p_{(r)} = m$, and $q_{(r)} = s$. One can easily check that $|T| = r^4$, so to compute (2) requires $\mathcal{O}(r^4)$ time. A more efficient way is to compute the sum

$$C'_{ms} = \sum_{(i,j,k,l) \in T'} a_{p(i)p(j)p(k)p(l)} b_{q(i)q(j)q(k)q(l)}, \tag{3}$$

where

$$T' = \{(i, j, k, l) \mid i, j, k, l = 1, 2, \ldots, r, \text{ and } \{i, j, k, l\} \cap \{r\} \neq \emptyset\},$$

$p_{(r)} = m$, and $q_{(r)} = s$. Since $|T'| = \binom{4}{1}(r-1)^3 + \binom{4}{2}(r-1)^2 + \binom{4}{3}(r-1) + 1$, then each sum C'_{ms} can be calculated in $\mathcal{O}(r^3)$ time. T' contains the quadruples of all combinations of indices that correspond to the order of the already-made assignments, having at least one occurrence of the index r. Note that $C'_{ms} \leq C_{ms}$ and the difference $C_{ms} - C'_{ms}$ is the same for all possible assignments $m \rightarrow s$, which implies that that the smallest C'_{ms} corresponds to the smallest C_{ms}.

The greedy function in Stage 2 selects the assignment that has the minimum cost of interaction with respect to the already-made assignments, i.e. the assignment m to s that has the smallest sum C_{ms}. Consider a GRASP iteration and let α be the RCL parameter defined in Stage 1 of the construction phase. The U costs are sorted and the smallest $\lfloor \alpha U \rfloor$ are placed in RCL. One assignment (say m to s) is selected at random from the RCL and the set Γ_r is updated, i.e.

$$\Gamma_r = \Gamma_r \cup \{(m, s)\}.$$

Stage 2 of the construction phase is presented in pseudo-code in Figure 2.

Procedure `ConstructStage2` takes as input the RCL parameter α and the four initial assignments made in Stage 1 and makes the remaining $n - 4$ assignments. In lines 4 to 12, the costs (C_{ms}) of all unassigned pairs are computed

```
procedure ConstructionStage2(α, (j1, p1), (j2, p2), (j3, p3), (j4, p4))
1     Γ = {(j1, p1), (j2, p2), (j3, p3), (j4, p4)};
2     do r = 5, ..., n − 1  →
3          U = 0;
4          do m = 1, ..., n →
5               do s = 1, ..., n →
6                    if (m, s) ∉ Γ →
7                         C_ms = Σ_{(i,j,k,l)∈T'} a_{p(i)p(j)p(k)p(l)} b_{q(i)q(j)q(k)q(l)};
8                         inheap(C_ms);
9                         U = U + 1;
10                   fi;
11              od;
12         od;
13         t =random[1, ⌊αU⌋];
14         do i = 1, ..., t →
15              C_ms =outheap();
16         od;
17         Γ = Γ ∪ {(m, s)};
18    od;
19    return Γ
end ConstructStage2;
```

Figure 2: Pseudo-code for Stage 2 of GRASP construction phase

and inserted into the heap to be sorted. Procedure `inheap` insert an element into the head and updates the heap. In lines 13 to 16, a number t is randomly generated in the interval $[1, \lfloor \alpha U \rfloor]$ and the t-th smallest cost is retrieved from the heap. The procedure `outheap` deletes the smallest element from the heap and updates the heap. In line 17 the set Γ of already-made assignments is updated. The algorithm continues its iterations until $|\Gamma| = n - 1$, in which case the solution is completed by adding the n-th remaining assignment.

2.3 Local Search Phase

The second phase of each GRASP iteration is local search. Given a neighborhood definition, the neighborhood $N(s)$ of a solution s is searched in an attempt to identify a better solution s'. If a better solution s' exists in $N(s)$, the current solution is replaced by s'. The procedure is repeated until no further improvement is possible.

In the implementation described in this paper, the local search implemented is the 2-exchange, similar to the local search for the QAP used in [6, 7, 8, 9]. We next describe k-exchange, a generalization of 2-exchange.

Let the difference between two permutations p and q be $\delta(p, q) = \{i \mid p(i) \neq q(i)\}$, and define the distance between the two permutations to be $d(p, q) = |\delta(p, q)|$. The k-exchange neighborhood $N_k(p)$ for a permutation $p \in \Pi_N$ is

$$N_k(p) = \{q \mid d(p, q) \leq k, 2 \leq k \leq n\}.$$

Pseudo-code for the k-exchange local search is shown in Figure 3.

The size of the neighborhood used in the k-exchange local search is $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. Let p be the starting permutation. The algorithm examines the set of permutations obtained by interchanging k elements of p. In line 3 of the pseudo-code, procedure `ProducePermutation`$(q, m, N_k(p))$ generates a permutation q by exchanging k elements of the current permutation p. Which elements are selected for exchange is a function of the value of m where $1 \leq m \leq \binom{n}{k}$. Once a neighboring permutation that improves the objective function is found (line 6), the current solution is updated and the procedure is restarted, searching the new neighborhood. This is repeated until no further improvement is possible. This type of search is called *first decrement* search. Another type, called *complete enumeration*, examines all possible solutions that can be generated by the exchange of k elements, i.e. examines the complete set of $\binom{n}{k}$ exchanges in the neighborhood and picks the best [8].

In our implementation, we choose first decrement 2-exchange search. The neighborhood $N_2(p)$ of a fixed permutation p consists of all those permutations that differ from p in exactly two places, i.e., that can be obtained by applying a transposition to p.

As we see in line 6 of the pseudo-code, the calculation of the objective function value of a permutation $q \in N(p)$ where p is the current permutation,

```
procedure procedure kexchange(k, p)
1    StopFlag=false;
2    do StopFlag=false →
3        StopFlag=true;
4        do m = 1, ..., (n k) →
5            ProducePermutation(q, m, N_k(p));
6            if cost(q) < cost (p) →
7                p = q;
8                StopFlag=false;
9                break;
10           fi;
11       od;
12   od;
13   return p
end kexchange;
```

Figure 3: Pseudo-code of k-exchange local search

is repeated at each iteration of the local search. In the worst case, there are $\binom{n}{2}$ repetitions, i.e. as many as the size of the neighborhood. Since there may be numerous repetitions, it is desirable to have an efficient procedure to decide if permutation q is better than permutation p. Denote by $Z(q)$ the objective function value of a permutation q. A straightforward evaluation of the gain $\Delta Z = Z(p) - Z(q)$ requires $\mathcal{O}(n^4)$ time, since the computation of $Z(q)$, for any $q \in \Pi_N$, takes $\mathcal{O}(n^4)$ time.

However, as shown in [1], ΔZ may be computed in $\mathcal{O}(n^3)$ time, using the following procedure. Suppose that $d(p,q) = 2$, i.e.

$$p(k_o) = q(l_o) \text{ and } p(l_o) = q(k_o).$$

To compute the gain $\Delta Z = Z(p) - Z(q)$, we evaluate the sum

$$\Delta Z = \sum_{(i,j,k,l) \in K_2} a_{ijkl}(b_{p(i)p(j)p(k)p(l)} - b_{q(i)q(j)q(k)q(l)})$$

where

$$K_2 = \{(i,j,k,l) \mid i,j,k,l = 1,2,\ldots,n, \{i,j,k,l\} \cap \{k_o, l_o\} \neq \emptyset\}.$$

Since $|K_2| = 2^1\binom{4}{1}(n-2)^3 + 2^2\binom{4}{2}(n-2)^2 + 2^3\binom{4}{3}(n-2)^1 + 2^4\binom{4}{4}(n-2)^0$, then it follows that the gain can be calculated in $\mathcal{O}(n^3)$ time.

2.4 Relating GRASP for QAP, BiQAP, and N-adic assignment problem

As aforementioned, the QAP is a special case of the BiQAP. It is natural to assume then, that the GRASP for BiQAP can be reduced to the GRASP for QAP described in [6]. If we consider Stage 1 of the construction phase as applied to a QAP of size n, we must make the two initial assignments from the available n, each with an interaction cost

$$C_{\mathcal{A}} = \sum_{i=1}^{2} \sum_{j=1}^{2} a_{p_{\mathcal{A}}(i)p_{\mathcal{A}}(j)} b_{q_{\mathcal{A}}(i)q_{\mathcal{A}}(j)}$$

as defined in (1), where a and b are $n \times n$ arrays and the permutations $p_{\mathcal{A}}, q_{\mathcal{A}} \in K = \{(i,j) \mid i,j = 1,2,\ldots,n, i \neq j\}$. Note that $|K| = n(n-1)$ and the number of feasible sets of two assignments is $|K|^2$. However, for a *symmetric* QAP, the cost for each pair of assignments reduces to

$$C_{\mathcal{A}} = 2a_{p_{\mathcal{A}}(1)p_{\mathcal{A}}(2)} b_{q_{\mathcal{A}}(1)q_{\mathcal{A}}(2)}.$$

One way to compute the sum $C_{\mathcal{A}}$ is to sort the nondiagonal elements of a in increasing order and of b in decreasing order, and then sort their corresponding products in increasing order. This will produce a set of the smallest $C_{\mathcal{A}}$ values that constitutes the RCL, exactly as defined in [6].

For Stage 2 of the construction phase, following the definition in (3), the cost of making the r-th assignment (of m to s) for a QAP is

$$C_{ms} = \sum_{(i,j) \in T'} a_{p(i)p(j)} b_{q(i)q(j)}, \tag{4}$$

where

$$T' = \{(i,j) \mid i,j = 1,2,\ldots,r, \{i,j\} \cap \{r\} \neq \emptyset\}.$$

Considering the fact that the QAP is symmetric and that $p(r) = m$ and $q(r) = s$, the cost in (4) can be rewritten as

$$C_{ms} = 2 \sum_{i=1}^{r-1} a_{mp(j)} b_{sq(j)}, \tag{5}$$

which is how the Stage 2 costs are computed in the GRASP for QAP of [6].

Finally, the local search procedure used in [6] is a 2-exchange neighborhood search where the gain from exchanging k_o with l_o in a given solution is computed in $\mathcal{O}(n)$ time. More specifically, given two permutations p and q that differ in only two positions, i.e.

$$p(k_o) = q(l_o) \text{ and } p(l_o) = q(k_o),$$

```
12 150 MHZ IP19 Processors
CPU: MIPS R4400 Processor Chip Revision:  5.0
FPU: MIPS R4010 Floating Point Chip Revision:  0.0
Data cache size:  16 Kbytes
Instruction cache size:  16 Kbytes
Secondary unified instruction/data cache size:  1 Mbyte
Main memory size:  2560 Mbytes, 2-way interleaved
```

Figure 4: Hardware configuration (partial output of system command `hinv`)

we can compute their difference in the objective function cost by evaluating the summation

$$\sum_{(i,j)\in K_2} a_{ij}(b_{p(i)p(j)} - b_{q(i)q(j)}),$$

where

$$K_2 = \{(i,j) \mid i,j = 1,2,\ldots,n, \{i,j\} \cap \{k_o,l_o\} \neq \emptyset\}.$$

The above analysis shows that it is very natural to extend the GRASP for QAP to a GRASP for BiQAP. In a similar fashion, GRASP can be extended to solve instances of the n-adic assignment problem [5] (e.g. for $n = 3$ we have the 3-dimensional assignment problem).

3 Computational Results

In this section, we report on a computational experiment to evaluate the accuracy and efficiency of a Fortran 77 implementation of the GRASP described in Section 2.

The experiment was done on a 150MHz Silicon Graphics (SGI) Challenge computer, whose hardware configuration is summarized in Figure 4. The code was compiled on the SGI Fortran compiler `f77` using compiler flags `-O2 -Olimit 800 -static`. The SGI Irix 5.3 operating system was in use during the experiments. Though the machine used in the experiments is configured with 12 processors, processes were limited to run on at most one processor at a time. CPU times in seconds were computed by calling the system routine `etime`. Reported CPU times exclude problem input and output report times. The portable Fortran pseudo number generator of Schrage [10] was used.

The GRASP for BiQAP has several parameters that need to be set. We fixed these parameters throughout all runs in the experiment. The RCL parameters used were $\alpha = 0.25$, $\beta = 0.3$, and $\Lambda = 1000$. A maximum of 150 GRASP iterations were allowed per run.

Table 1: Statistics summarizing GRASP runs

			computational effort to find optimal permutation					
			iterations			CPU time		
problem	n	opt value	min	avg	max	min	avg	max
biQAP_10	10	133216	1	1.7	2	1.3	2.3	3.4
biQAP_12	12	120600	1	7.2	18	5.3	23.1	56.9
biQAP_14	14	251356	1	2.2	4	8.3	23.9	41.5
biQAP_16	16	441696	2	13.4	34	44.4	289.6	749.5
biQAP_18	18	811338	1	2.8	5	38.5	138.7	281.2
biQAP_20	20	1487296	1	2.0	5	74.8	184.2	468.1
biQAP_22	22	3118716	1	1.5	2	133.6	267.1	431.5
biQAP_24	24	1780378	1	28.2	66	509.7	8092.2	19047.7
biQAP_26	26	5305872	1	3.1	9	471.7	1664.3	4717.1
biQAP_28	28	4606532	1	4.7	15	568.8	4130.4	13683.0
biQAP_30	30	4468860	4	27.3	138	5370.5	30344.0	152500.1
biQAP_32	32	8088327	1	6.6	16	2644.5	12585.1	27822.8
biQAP_34	34	9445329	4	14.3	63	9922.9	36936.5	161505.3
biQAP_36	36	11297520	2	21.8	58	6102.4	79281.6	208924.6

To test the accuracy of the GRASP, i.e. how the solution found by GRASP compares with an optimal solution, we tested our code on 14 test problems having known optimal solutions, whose generator is described in [2]. In addition to the limit of 150 GRASP iterations, the algorithm stops if an optimal permutation is found. The instances varied in dimension from $n = 10$ to $n = 36$, the largest having 1,679,616 entries in each of its two matrices. These instances were also used by Burkard and Çela [1] to test several heuristics for the BiQAP. The GRASP uses two random number streams. The first is used to select the Λ initial assignments, while the second is used to build the restricted candidate lists during GRASP construction phase. In our experiment, ten replications of the GRASP runs were done for each test problem, each one using a different seed to generate the pseudo random number stream used to select the initial Λ assignments. The seeds used are $270001, 270002, \ldots, 270010$. For determining the RCL, the seed used is 12444.

For all 14 test problems considered, the GRASP found an optimal permutation on every single replication, in as few as one GRASP iteration, and in as many as 138 iterations. Table 1 summarizes the computational experiments. For each test problem instance, the table lists its name, dimension, cost of an optimal permutation, and the minimum, average, and maximum number of iterations and CPU time to find an optimal permutation. Figure 5 shows the distribution of GRASP iterations for all test problem replications. along with the average number of iterations taken. Figure 6 shows the distribution of CPU time together with average CPU time.

Our results indicate that the GRASP compares well with the heuristics described and tested in Burkard and Çela [1]. That paper investigated the perfor-

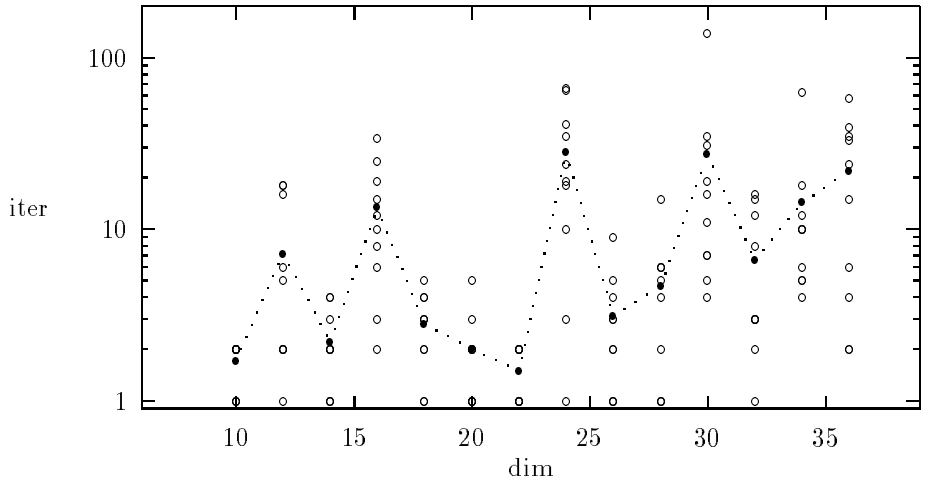

Figure 5: Number of GRASP iterations to find optimal solution as a function of the dimension of the BiQAP. Dotted line is average number of iterations. All instances included.

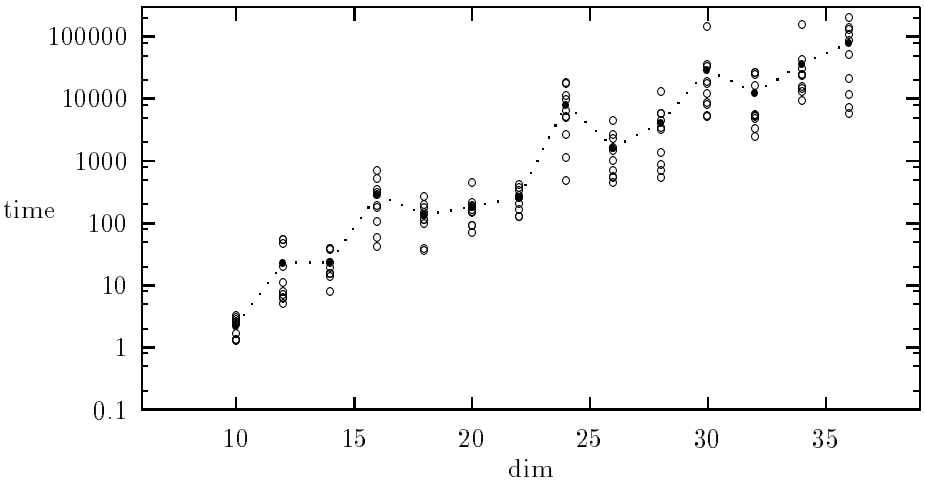

Figure 6: GRASP time (in seconds) to find optimal solution as a function of the dimension of the BiQAP. Dotted line is average solution time. All instances included.

mance of Heider's method [4], the first and best improvement methods, three versions of simulated annealing, two versions of tabu search and a combination of tabu search with simulated annealing. In that paper, each heuristic is repeatedly tested on seven BiQAP instances of dimensions $n = 10, 14, 16, 20, 24, 28, 32$. Additionally, Heider's method, and the first and best improvement heuristics were tested on an instance of size $n = 36$. The third version of simulated annealing (SimAnn3) proved to be the best among all heuristics tested by Burkard and Çela. SimAnn3 solved almost all of the instances of sizes $n = 14, 16, 20$, while the average percentage of optimally solved instances for the remaining problems is about 46%. None of the nine heuristics tested by Burkard and Çela solved the entire set of instances of dimension $n \geq 12$ to optimality.

Although our GRASP code solved all ten replications of all 14 test problems, it often required a large amount of CPU time to find an optimal permutation. The longest run in our experiment took over two days of CPU time. This can, however, be remedied with distributed parallel computing. Suppose we have ten processors to run concurrently, and allocate to each processor a copy of the code, a copy of the data, and a different random number seed to generate the random number stream used to select the initial Λ assignment, stopping when the first processor finds the optimal permutation. The running time of the 10-processor system would correspond to the minimum CPU time in Table 1. Those range from 1.3 seconds for the $n = 10$ problem, to a little under three hours of CPU time for the $n = 34$ instance. The largest ($n = 36$) test problem would be solved in less than two hours.

On the 14 instances considered in this experiment, 10 were solved in a minimum of one GRASP iteration, two in a minimum of two iterations, and two in a minimum of four iterations. It is easy to see that no matter how many processors are used in a distributed parallel implementation of GRASP of the kind described above, the GRASP running time is bounded below by the running time of one GRASP iteration. Figure 7 shows, for the 14 test problems used in the experiment, average CPU time per GRASP iteration, as a function of BiQAP size, as well as estimated running times for larger instances. The figure suggests that one GRASP iteration on an instance of dimension $n = 90$ would take over 10 CPU days.

Finally, it is interesting to note that for each instance, all GRASP replications produced identical optimal permutations, suggesting that these solutions are unique. Table 2 lists the optimal permutations found.

4 Concluding Remarks

In this paper we describe a GRASP for finding approximate solutions of the biquadratic assignment problem and test a Fortran 77 implementation of the algorithm on a set of test problems with known optimal solutions. Computational experience with the code indicates that the GRASP is effective in finding

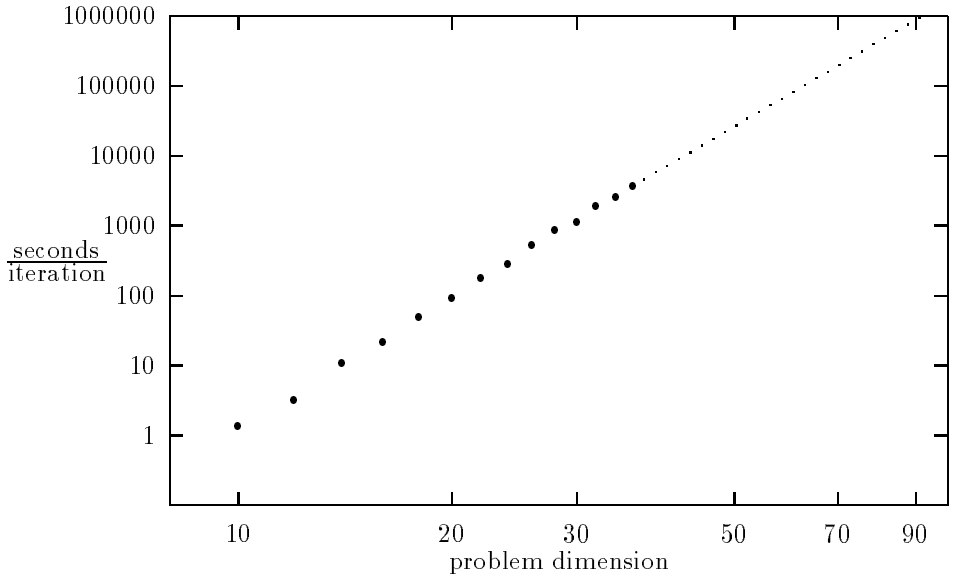

Figure 7: CPU time per GRASP iteration as a function of BiQAP dimension. Computed values for $n \leq 36$. Estimated time per iteration for $n > 36$.

the optimal permutation for these test problems. On all test problems considered, the optimal solution was found on all ten runs, each starting with a different seed for the pseudo random number generator.

Solution times can be long on the larger problems, with a single GRASP iteration requiring, for an instance of dimension $n = 36$, as much as one hour of CPU time on a SGI Challenge computer. To mitigate this problem, we suggest three directions. Firstly, a version of the code that takes advantage of sparsity in the problem matrices can be developed, as was done for the GRASP for QAP [7]. This could make it easier to solve large sparse problems that occur in practice. Secondly, as we discussed in Section 3, a distributed parallel GRASP could be implemented, as was done in [8] for the QAP. Finally, it may be possible to parallelize parts of the computation in each GRASP iteration.

Acknowledgement

The authors would like to thank R. Burkard and E. Çela for providing the test data used in the computational experiments.

Table 2: Optimal permutations found by all replications of GRASP

n	optimal permutation
10	10,9,1,2,3,4,5,6,7,8
12	12,11,1,2,3,4,5,6,7,8,9,10
14	14,13,1,2,3,4,5,6,7,8,9,10,11,12
16	16,15,1,2,3,4,5,6,7,8,9,10,11,12,13,14
18	18,17,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
20	20,19,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
22	22,21,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
24	7,6,9,17,4,15,22,16,8,13,21,5,1,3,12,24,20,10,14,19,18,11,23
26	26,25,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24
28	24,14,5,10,1,7,4,26,2,17,23,15,8,6,21,3,27,9,20,16,22,11,13,25,18,28,12,19
30	17,15,14,18,4,27,30,16,23,19,24,7,11,8,21,9,3,28,22,29,6,20,10,12,26,13,25,2,5,1
32	2,22,11,29,32,16,14,7,20,17,18,9,31,19,5,15,21,12,4,6,8,28,30,10,1,24,25,26,23,3,27,13
34	26,10,5,21,34,16,19,9,22,31,24,3,30,27,14,29,4,7,17,25,28,1,18,33,2,6,32,8,12,15,13,20,11
36	26,36,11,24,8,21,33,32,14,31,5,22,16,2,18,9,29,35,20,6,7,27,19,17,10,25,34,1,30,12,15,13,4,28,3,23

References

[1] R. Burkard and E. Çela, *Heuristics for biquadratic assignment problems and their computational comparison*, European Journal of Operational Research, 83 (1995), pp. 283–300.

[2] R. Burkard, E. Çela, and B. Klinz, *On the biquadratic assignment problem*, in Quadratic assignment and related problems, P. Pardalos and H. Wolkowicz, eds., vol. 16 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1994, pp. 117–146.

[3] T. Feo and M. Resende, *Greedy randomized adaptive search procedures*, Journal of Global Optimization, 16 (1995), pp. 109–133.

[4] C. Heider, *A computationally simplified pair exchange algorithm for the quadratic assignment problem*, Tech. Rep. 101, Center for Naval Analyses, Arlington, VA, 1972.

[5] E. Lawler, *The quadratic assignment problem*, Management Science, 9 (1963), pp. 586–599.

[6] Y. Li, P. Pardalos, and M. Resende, *A greedy randomized adaptive search procedure for the quadratic assignment problem*, in Quadratic assignment and related problems, P. Pardalos and H. Wolkowicz, eds., vol. 16 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1994, pp. 237–261.

[7] P. Pardalos, L. Pitsoulis, and M. Resende, *Fortran subroutines for approximate solution of sparse quadratic assignment problems using GRASP*, tech. rep., AT&T Bell Laboratories, Murray Hill, NJ, 1995.

[8] ——, *A parallel GRASP implementation for the quadratic assignment problem*, in Solving Irregular Problems in Parallel: State of the Art, A. Ferreira and J. Rolim, eds., Kluwer Academic Publishers, 1995, pp. 111–130.

[9] M. Resende, P. Pardalos, and Y. Li, *Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP*, ACM Transactions on Mathematical Software, 22 (1996), pp. 104–118.

[10] L. Schrage, *A more portable Fortran random number generator*, ACM Transactions on Mathematical Software, 5 (1979), pp. 132–138.