Algorithm ____ : FORTRAN Subroutines for Approximate Solution of Dense Quadratic Assignment Problems using GRASP

MAURICIO G.C. RESENDE
AT&T Bell Laboratories
PANOS M. PARDALOS
University of Florida
and
YONG LI
Pennsylvania State University

In the NP-complete quadratic assignment problem (QAP), n facilities are to be assigned to n sites at minimum cost. The contribution of assigning facility i to site k and facility j to site l to the total cost is $f_{ij} \cdot d_{kl}$, where f_{ij} is the flow between facilites i and j, and d_{kl} is the distance between sites k and l. Only very small ($n \leq 20$) instances of the QAP have been solved exactly, and heuristics are therefore used to produce approximate solutions. This paper describes a set of FORTRAN subroutines to find approximate solutions to dense quadratic assignment problems, having at least one symmetric flow or distance matrix. A greedy randomized adaptive search procedure (GRASP) is used to produce the solutions. The design and implementation of the code are described in detail, and extensive computational experiments are reported, illustrating solution quality as a function of running time.

Categories and Subject Descriptors: G.1.6 [**Numerical Analysis**]: Optimization—*integer programming*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms*; G.m [**Mathematics of Computing**]: Miscellaneous—*FORTRAN*

General terms: Algorithms, Performance, FORTRAN
Additional Key Words and Phrases: Combinatorial optimization, quadratic assignment problem, local search, GRASP, FORTRAN subroutines

Authors' addresses: M.G.C. Resende, Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ; P.M. Pardalos, Department of Industrial and Systems Engineering, The University of Florida, Gainesville, FL; Y. Li, Department of Computer Science, The Pennsylvania State University, University Park, PA.

1. INTRODUCTION

Given a set $\mathcal{N} = \{1, 2, \ldots, n\}$ and $n \times n$ matrices $F = (f_{ij})$ and $D = (d_{kl})$, the quadratic assignment problem (QAP) can be stated as follows:

$$\min_{p \in \Pi_{\mathcal{N}}} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)},$$

where $\Pi_{\mathcal{N}}$ is the set of all permutations of $\mathcal{N}$. One of the major applications of the QAP is in location theory where the matrix $F = (f_{ij})$ is the flow matrix, i.e. f_{ij} is the flow of materials from facility i to facility j, and $D = (d_{kl})$ is the distance matrix, i.e. d_{kl} represents the distance from location k to location l [2, 3, 5]. The contribution to the total cost of simultaneously assigning facility i to location k and facility j to location l is $f_{ij} \cdot d_{kl}$. The objective is to find an assignment of all facilities to all locations (i.e. a permutation $p \in \Pi_{\mathcal{N}}$), such that the total cost of the assignment is minimized. Throughout this paper we often refer to the QAP in the context of this location problem. For a survey of the QAP, see [7].

Li, Pardalos and Resende [6] describe a greedy randomized adaptive search procedure (GRASP) for finding approximate solutions of dense quadratic assignment problems. The algorithm was extensively tested on QAPLIB [1], a standard set of benchmark quadratic assignment problems, producing best known solutions for most problems in the suite. In this paper, we describe an optimized version of the implementation used in [6]. This implementation has a limitation that matrices F and D must be symmetric. This limitation, in practice, only requires that one of the two matrices be symmetric, due to the fact that for any QAP with one symmetric matrix, there exists a corresponding QAP where both matrices are symmetric, possesing the same optimal permutations, with the value of the optimal solution scaled up by a factor of two. The new QAP is defined by the original symmetric matrix and the sum of the original unsymmetric matrix and its transpose.

GRASP is an iterative sampling method for combinatorial optimization [4]. A number of GRASP iterations are carried out, each iteration producing an approximate solution to the optimization problem. The best solution over all iterations is returned by the algorithm as the GRASP solution. Each iteration is made up of two phases: a construction phase and a local search phase. In the construction phase, a solution is constructed, guided by a greedy function. Due to the randomization employed, the solution is not necessarily greedy. In the local search phase, the neighborhood around the constructed solution is searched for an improved solution.

In this GRASP for QAP, the construction phase has two stages. In stage 1, two assignments are produced, i.e. facility i is assigned to site k and facility j is assigned to site l. The idea is to assign facilities with high interaction, i.e. having high f_{ij} values, to nearby sites, i.e. sites with low d_{kl} values. To do this, the procedure sorts inter-site distances in increasing order and inter-facility flows in decreasing order. Let $d_{k_1,l_1} \leq d_{k_2,l_2} \leq \cdots \leq d_{k_p,l_p}$ and $f_{i_1,j_1} \geq f_{i_2,j_2} \geq \cdots \geq f_{i_p,j_p}$ be the sorted values, where $p = n^2 - n$. The products $d_{k_1,l_1} \cdot f_{i_1,j_1}, d_{k_2,l_2} \cdot f_{i_2,j_2}, \ldots, d_{k_p,l_p} \cdot f_{i_p,j_p}$ are then sorted in increasing order. Among the smallest $d_{kl} \cdot f_{ij}$ products, one, corresponding to the pair of stage 1 assignments, is selected at random. Sorting all of the $p = n^2 - n$ distances and flows is inefficient and offers little benefit. Instead, only the best $n_\beta = \beta p$ values are sorted, where β is a parameter such that

$0 < \beta \leq 1$. Among these n_β pairs of assignments, a pair is selected at random from the set of αn_β assignments having the smallest $d_{kl} \cdot f_{ij}$ products, where α is such that $0 < \alpha \leq 1$.

In stage 2 of the construction phase, the remaining $n-2$ facility-site assignments are made sequentially. The idea is to favor assignments that have small interaction cost with the set of previously-made assignments. Let Γ be the set of q assignments at a given point in the construction phase, i.e. $\Gamma = \{(i_1, k_1), (i_2, k_2), \ldots, (i_q, k_q)\}$. The cost of assigning facility j to site l, with respect to the already-made assignments, is defined to be

$$c_{jl} = \sum_{(i,k)\in\Gamma} f_{ij} d_{kl}.$$

All costs of unassigned facility-site pairs (j, l) are sorted in increasing order. Of the pairs having the least $\alpha \cdot |\Gamma|$ costs, one is selected at random and is added to the set Γ. The procedure is repeated until $n-1$ assignments are made. The remaining facility is then assigned to the remaining site.

In the local search phase of this GRASP, a 2-exchange neighborhood search is conducted on the constructed solution. There, all possible 2-swaps of facility-locations are considered. If a swap improves the cost of the assignment, it is accepted. The procedure continues until no swap improves the solution value.

The paper is organized as follows. In Section 2 we describe the design and implementation of the set of FORTRAN subroutines distributed with the package. Usage is described in Section 3. Computational testing is presented in Section 4 and concluding remarks are made in Section 5.

2. DESIGN AND IMPLEMENTATION

We followed the following design guidelines in the implementation of the set of subroutines. The code is written in ANSI standard FORTRAN 77 and is intended to run without modification on UNIX platforms (it should run on other environments without modification). There are no `common` blocks in the code and all arrays and variables are passed by parameter. The optimizer is a self-contained set of subroutines. Input and output, as well as array declarations and parameter settings, are done independently, outside of the optimizer module.

The distribution consists of three files: `Makefile`, `driver.f`, and `gqapd.f`. The `Makefile` is used to produce an executable `gqapd` in a UNIX environment. The file `driver.f` defines all of the arrays and parameters to be used by the GRASP subroutine `gqapd`, inputs the problem, calls the GRASP, and outputs the solution. The file `gqapd.f` is the core of the package, with the subroutines that make up the optimizer.

The following modules make up the package:

– `Makefile`: A makefile to compile and link the supplied driver with the subroutine package producing the executable `gqapd`.
– `program driver`: An example of a driver for the optimizer. The driver includes a subroutine to input QAPLIB instances (`subroutine readp`) and a subroutine to output the GRASP solution (`subroutine outsol`).
Functions and/or subroutines called: `readp`, `gqapd`, and `outsol`.
See Usage Notes.

- `subroutine gqapd`: Main subroutine to control the GRASP iterations.
  Functions and/or subroutines called: `srtcst`, `stage1`, `stage2`, and `local`.
  See Usage Notes.
- `subroutine srtcst`: Subroutine that sorts the costs $f_{ij} \cdot d_{kl}$ in increasing order.
  Functions and/or subroutines called: `insrtq` and `removq`.
- `subroutine stage1`: Subroutine implementing stage 1 of the GRASP construction phase. Builds two assignments for the construction phase (assigns facility i to site k and facility j to site l).
  Functions and/or subroutines called: `randp`.
- `subroutine stage2`: Subroutine implementing stage 2 of the GRASP construction phase. Builds a randomized greedy permutation starting from the assignments made in `subroutine stage1`. Returns permutation and objective function value.
  Functions and/or subroutines called: `insrtq`, `randp`, and `removq`.
- `subroutine savsol`: Saves incumbent solution.
  Functions and/or subroutines called: None.
- `subroutine local`: Local 2-exchange on permutation array produced in stage 2 of GRASP construction phase. Returns possibly improved permutation and objective function value.
  Functions and/or subroutines called: `mkbseq`, `evalij`, and `swapij`.
- `subroutine mkbseq`: Given two permutation arrays a and b, applies the same transformation to both, making $b = \{1, 2, \ldots, n\}$.
  Functions and/or subroutines called: None.
- `subroutine insrtq`: Inserts an element $\{v, iv\}$ into a queue $\{q, iq\}$.
  Functions and/or subroutines called: `upheap`.
- `subroutine upheap`: Updates heap to proper order.
  Functions and/or subroutines called: None.
- `subroutine removq`: Removes smallest element $\{v, iv\}$ from a priority queue $\{q, iq\}$.
  Functions and/or subroutines called: `dnheap`.
- `subroutine dnheap`: Updates heap to proper order.
  Functions and/or subroutines called: None.
- `real function randp`: Portable pseudo-random number generator [8]. Generates an integer number in the range $[0, 2^{31} - 1]$.
  Functions and/or subroutines called: None.
- `subroutine evalij`: Computes the gain in the objective function obtained by switching the locations of facilities i and j $(i < j)$.
  Functions and/or subroutines called: None.
- `subroutine swapij`: Swaps components i and j of an integer array.
  Functions and/or subroutines called: None.

Subroutine `gqapd` takes as input the input data (`n`, `f`, `d`), GRASP parameters (number of iterations (`maxitr`), α, β, target value (`look4`), random number seed (`seed`), and a number of auxiliary arrays, and returns the best permutation found (`opta`), its cost (`bestv`), and number of GRASP iterations taken (`iter`). Before the GRASP iterations are executed, the value of the best solution found is initialized to a large value and the stage 1 costs $f_{ij} \cdot d_{kl}$ are sorted in subroutine `srtcst`.

The do loop 1010 iterates the GRASP. In each GRASP iteration, the initial two assignments are made in subroutine stage1. The construction phase is concluded with $n-2$ assignments in subroutine stage2, and the local search around the neighborhood of the constructed solution is carried out in subroutine local. If the best solution of the iteration (objv) is better than the incumbent (bestv), the incumbent is updated in subroutine savsol. If the target solution (look4) is found, that solution is returned by the GRASP.

Since stage 1 of the construction phase uses the same sorted assignment costs during each GRASP iteration, the costs are sorted once, outside the main GRASP loop, in subroutine srtcst. The subroutine uses heap sort to partially sort the flow values, distance values, and assignment costs. The do loops 1010 and 1020 insert off-diagonal distance and flow values into their respective priority heaps. Parameter n_β is computed and in do loop 1030 the smallest distance value and largest flow values are removed from the heaps, the assignment cost is computed and inserted into its priority heap. The do loop is repeated n_β times. The do loop 1040 puts into array cost the n_β smallest assignment costs by sequentially removing from the assignment cost priority heap the smallest cost element. Information needed to retrieve the assignments corresponding to each cost is put in the array fdind.

Subroutine stage1 implements stage 1 of the GRASP construction phase. The permutation arrays a and b are initialized, the index (nselct) of the assignment pair is chosen at random, and the assignment indices i, j, k, l are recovered from the data structure. Just prior to do loop 1020, the first assignment is put in the permutation arrays and in do loops 1020 and 1030 the second assignment is put in the arrays.

Subroutine stage2 implements stage 2 of the GRASP construction phase. The do loop 1020 makes assignments $3, 4, \ldots, n-1$. The cost, with respect to previously-assigned pairs, of each possible assignment is computed in loops 1030, 1040, and 1050. The cost is inserted in its priority heap for sorting. After all cost have been computed, the index (nselct) of the randomly selected assignment is determined and the assignment is retrieved from the heap in do loop 1070. Permutation arrays a and b are updated with the latest assignment in do loops 1073 and 1074 and at the end of do loop 1020. The cost of the last assignment is added to the total assignment cost in do loop 2050.

Subroutine local implements the 2-exchange of the GRASP local search phase. The local search is carried out on permutation array a, so a rearrangement is done in subroutine mkbseq to make permutation array $\mathtt{b} = \{1, 2, \ldots, n\}$. In do loops 1020 and 1030, for all pairs i, j such that $\mathtt{j} > \mathtt{i}$, subroutine evalij evaluates the gain xgain of swapping i and j in permutation array a. If a positive gain is attained, the swap is carried out in subroutine swapij and the cost of the assignment (objv) is updated. The procedure ends when no further improvement is possible by swapping elements in the permutation array.

Subroutine savsol saves permutation a in array opta and the cost of the assignment (objv) in scalar bestv.

In subroutine insrtq, a pair of elements (v and iv) is inserted into a priority heap of pairs (q and iq), ordered by the values in q. The size of the heaps (sizeq) is updated. Subroutine upheap updates the heaps, to take into account the additional elements.

Given two arrays of elements `q` and `iq`, such that, except for the first element, `q` is sorted as a heap, subroutine `upheap` orders the arrays identically, such that array `q` becomes completely sorted as a heap.

In subroutine `removq`, a pair of elements (`v` and `iv`) is removed from a priority heap of pairs (`q` and `iq`), ordered by the values in `q`. The size of the heaps (`sizeq`) is updated. Subroutine `dnheap` updates the heaps, to take into account the removal of the elements.

Given two arrays of elements `q` and `iq`, sorted as a heap, except for the first element, which has been removed, subroutine `dnheap` orders the arrays identically, such that array `q` becomes a priority heap.

Subroutine `randp` is the portable random number generator of Schrage [8]. Given a seed `ix` $\in [0, 2^{31} - 1]$, it returns by parameter a new seed in the same range and by value a real number in the interval [0,1].

Subroutine `evalij` evaluates the assignment cost gain attained by swapping indices `i` and `j` in an assignment permutation array `a`. Index `j` is assumed to be greater than `i`. The `do` loop 1010 computes the interaction with facilities having index less than `i`. The `do` loop 2010 computes the interaction with facilities having index greater than `i` but less than `j`. The `do` loop 3010 computes the interaction with facilities having index greater than `j`.

Subroutine `mkbseq` applies identical transformations to arrays `a` and `b` to make `b` $= \{1, 2, \ldots, n\}$.

Subroutine `swapij` swaps elements of array `x` indexed by `i` and `j`.

3. USAGE

The subroutines in file `gqapd.f` carry out the approximate optimization of the QAP. The user interface with them is subroutine `gqapd`. It must be called from a driver program. Subroutine `gqapd` takes as parameters the following:

- variables needed for input:
  - `n`: dimension of QAP (`integer*4`)
  - `nmax`: maximum dimension of QAP (`integer*4`)
  - `maxitr`: maximum number of GRASP iterations (`integer*4`)
  - `alpha`: GRASP construction phase parameter α (`real`)
  - `beta`: GRASP construction phase parameter β (`real`)
  - `look4`: GRASP returns permutation if solution with cost less than or equal to `look4` is found (`look4` $= -1$ causes GRASP to take `maxitr` iterations) (`integer*4`)
  - `seed`: seed for random number generator $\in [0, 2^{31} - 1]$ (`integer*4`)
- `integer*4` arrays needed for input:
  - `f`: flow matrix (represented as row-by-row array of dimension `nmax*nmax`)
  - `d`: distance matrix (represented as row-by-row array of dimension `nmax*nmax`)
- `integer*4` arrays needed for work:
  - `a`: dimension `nmax`
  - `b`: dimension `nmax`
  - `optb`: dimension `nmax`
  - `srtf`: dimension `nmax*nmax`
  - `srtif`: dimension `nmax*nmax`

```
12

0 1 2 3 1 2 3 4 2 3 4 5
1 0 1 2 2 1 2 3 3 2 3 4
2 1 0 1 3 2 1 2 4 3 2 3
3 2 1 0 4 3 2 1 5 4 3 2
1 2 3 4 0 1 2 3 1 2 3 4
2 1 2 3 1 0 1 2 2 1 2 3
3 2 1 2 2 1 0 1 3 2 1 2
4 3 2 1 3 2 1 0 4 3 2 1
2 3 4 5 1 2 3 4 0 1 2 3
3 2 3 4 2 1 2 3 1 0 1 2
4 3 2 3 3 2 1 2 2 1 0 1
5 4 3 2 4 3 2 1 3 2 1 0

0  5  2  4  1  0  0  6  2  1  1  1
5  0  3  0  2  2  2  0  4  5  0  0
2  3  0  0  0  0  0  5  5  2  2  2
4  0  0  0  5  2  2 10  0  0  5  5
1  2  0  5  0 10  0  0  0  5  1  1
0  2  0  2 10  0  5  1  1  5  4  0
0  2  0  2  0  5  0 10  5  2  3  3
6  0  5 10  0  1 10  0  0  0  5  0
2  4  5  0  0  1  5  0  0  0 10 10
1  5  2  0  5  5  2  0  0  0  5  0
1  0  2  5  1  4  3  5 10  5  0  2
1  0  2  5  1  0  3  0 10  0  2  0
```

Fig. 1. NUG12 QAPLIB instance

  - `srtd`: dimension `nmax*nmax`
  - `srtid`: dimension `nmax*nmax`
  - `srtc`: dimension `nmax*nmax`
  - `srtic`: dimension `nmax*nmax`
  - `indexd`: dimension `nmax*nmax`
  - `indexf`: dimension `nmax*nmax`
  - `cost`: dimension `nmax*nmax`
  - `fdind`: dimension `nmax*nmax`
- `integer*4` array needed for output:
  - `perm`: permutation vector of best solution found (dimension `nmax`)
- `integer*4` variables needed for output:
  - `bestv`: cost of best assignment found
  - `iter`: number of GRASP iterations taken

The sample driver program for `gqapd` included in the distribuition, is set for problems of dimension $n \leq 256$. All variables and arrays needed by subroutine `gqapd` are defined. Variable `iseed0`, used by the driver, is also defined. Subroutines `readp` and `outsol` are examples of code that can be used for input and output, respectively.

As an example, consider the QAP instance NUG12 with QAPLIB input file shown in Figure 1. Running the driver program on that input data produces the output

```
-----------------------------------------------------------
G R A S P for Q A P-------------------------------------

input---------------------------------------------------
dimension of qap                  :          12
construction phase parameter alpha :   0.2500000
construction phase parameter beta  :   0.5000000
maximum number of grasp iterations :         100
look4                             :          -1
initial seed                      :      270001

output--------------------------------------------------
grasp iterations                  :         100
cost of best permutation found    :         578
best permutation found            :    3   9   7  12   1
                                  :   11   8   4   2  10
                                  :    6   5

-----------------------------------------------------------
```

Fig. 2. Sample output of driver for QAP instance NUG12

shown in Figure 2.

4. COMPUTATIONAL RESULTS

In this section, we summarize the battery of runs carried out to test the algorithm. The experiments were carried out on a Silicon Graphics Challenge computer (150 MHz MIPS R4400 processor), with enough main memory so that swapping was never necessary. The code was compiled with the `f77` compiler using flags `-O2 -Olimit 800`. User times are reported, reflecting subroutine `gqapd` only (input and output times are not included) and are timed with the system subroutine `etime`. The code is tested on the suite of QAP test problems QAPLIB [1]. The GRASP was run on the 95 instances in the library of dimension $n \geq 8$ that are pure quadratic assignment problems, and have at least one symmetric distance or flow matrix. For each instance, eight settings for number of iterations were run (16, 32, 64, 128, 256, 512, 1024, and 2048). For each iteration setting, five runs were made with different random number generator seeds (1, 2, 3, 4, and 5). A total of 3,800 runs were made. The other parameters of the GRASP were set fixed throughout the experiment ($\alpha = .25$, $\beta = .5$, `look4` $= -1$). Tables I–III list the instances considered, with the best known solution, and the minimum, average, and maximum solutions found by the GRASP, over the five replications, for 32 and 2048 iteration runs.

Of the 3,800 runs, the algorithm produced the best known solution in 1,759 runs, was within 0.5% of the best known solution in 1,898 runs, and was within 1% in 2,443 runs. Figure 3 illustrates the quality of the solutions produced, as a function of problem dimension and number of GRASP iterations. Runs finding a solution within 0.1% of the best known are not shown in the figure. Those are illustrated in Figure 4. That figure shows the 1,759, 139, and 545 runs for which the algorithm matched the best known solution (bks), was between 0 and .5% of the bks, and was

		32 iterations			2048 iterations		
name	bks	min	avg	max	min	avg	max
chr12a	9552	9552	9902	10214	9552	9552	9552
chr12b	9742	9742	9814	10102	9742	9742	9742
chr12c	11156	11566	12033	12516	11156	11156	11156
chr15a	9896	10064	10594	10836	9896	9904	9936
chr15b	7990	8990	9287	9784	7990	7990	7990
chr15c	9504	10610	11908	13534	9504	9957	10446
esc08a	2	2	2	2	2	2	2
esc08b	8	8	8	8	8	8	8
esc08c	32	32	32	32	32	32	32
esc08d	6	6	6	6	6	6	6
esc08e	2	2	2	2	2	2	2
esc08f	18	18	18	18	18	18	18
esc16a	68	68	68	68	68	68	68
esc16b	292	292	292	292	292	292	292
esc16c	160	160	160	160	160	160	160
esc16d	16	16	16	16	16	16	16
esc16e	28	28	28	28	28	28	28
esc16f	0	0	0	0	0	0	0
esc16g	26	26	26	26	26	26	26
esc16h	996	996	996	996	996	996	996
esc16i	14	14	14	14	14	14	14
esc16j	8	8	8	8	8	8	8
lipa10a	473	473	473	473	473	473	473
lipa10b	2008	2008	2008	2008	2008	2008	2008
nug08	214	214	214	214	214	214	214
nug12	578	578	584	590	578	578	578
nug15	1150	1150	1157	1164	1150	1150	1150
rou10	174220	174220	174220	174220	174220	174220	174220
rou12	235528	235528	237935	240124	235528	235528	235528
rou15	354210	354210	361850	366930	354210	354210	354210
scr10	26992	26992	26992	26992	26992	26992	26992
scr12	31410	31410	31410	31410	31410	31410	31410

Table I. Summary of runs: best known solution (bks), minimum, average, and maximum GRASP solutions for 32 and 2048 iteration runs (problems of size $n \leq 16$)

		32 iterations			2048 iterations		
name	bks	min	avg	max	min	avg	max
chr18a	11098	13864	14356	14884	11098	11594	12404
chr18b	1534	1534	1594	1670	1534	1534	1534
chr20a	2192	2498	2594	2660	2224	2333	2406
chr20b	2298	2566	2718	2868	2462	2482	2516
chr20c	14142	14810	16805	18806	14142	14142	14142
chr22a	6156	6344	6458	6606	6320	6344	6368
chr22b	6194	6472	6564	6650	6300	6365	6410
chr25a	3796	4372	4618	4922	3884	4211	4344
els19	17212548	17212548	17714800	17937024	17212548	17212548	17212548
esc32a	130	144	146	150	134	134	136
esc32b	168	168	184	192	168	168	168
esc32c	642	642	642	642	642	642	642
esc32d	200	200	200	204	200	200	200
esc32e	2	2	2	2	2	2	2
esc32f	2	2	2	2	2	2	2
esc32g	6	6	6	6	6	6	6
esc32h	438	438	439	442	438	438	438
kra30a	88900	91700	92064	92470	88900	89924	90700
kra30b	91420	92220	92916	93940	91580	91686	91910
lipa20a	3683	3683	3720	3766	3683	3683	3683
lipa20b	27076	27076	27076	27076	27076	27076	27076
lipa30a	13178	13389	13399	13414	13178	13241	13343
lipa30b	151426	151426	156148	175037	151426	151426	151426
nug20	2570	2580	2598	2612	2570	2570	2570
nug30	6124	6162	6210	6248	6136	6152	6166
rou20	725522	735980	741357	745046	725662	728140	730290
scr20	110030	110968	112816	114426	110030	110036	110058
ste36a	9526	9830	9914	10018	9596	9661	9682
ste36b	15852	16160	16917	17414	15852	16049	16160
ste36c	8239110	8484420	8544170	8672034	8291830	8311120	8355466

Table II. Summary of runs: best known solution (bks), minimum, average, and maximum GRASP solutions for 32 and 2048 iteration runs (problems of size n, $18 \leq n \leq 36$)

		32 iterations			2048 iterations		
name	bks	min	avg	max	min	avg	max
esc128	64	64	64	64	64	64	64
esc64a	116	116	116	116	116	116	116
lipa40a	31538	31933	31949	31961	31884	31902	31911
lipa40b	476581	476581	493113	559240	476581	476581	476581
lipa50a	62093	62767	62790	62815	62704	62727	62747
lipa50b	1210244	1210244	1339350	1427509	1210244	1210244	1210244
lipa60a	107218	108211	108243	108262	108077	108148	108192
lipa60b	2520135	2520135	2902810	3000633	2520135	2520135	2520135
lipa70a	169755	171158	171185	171209	171047	171079	171102
lipa70b	4603200	5516027	5519179	5522184	4603200	4603200	4603200
lipa80a	253195	255066	255123	255176	254861	254941	254976
lipa80b	7783962	9363469	9379260	9398222	7763962	8715730	9360517
lipa90a	360630	363159	363204	363226	362917	362972	363028
lipa90b	12490441	15121756	15136900	15156071	12490441	12490441	12490441
sko100a	152002	153152	153767	154262	152926	153015	153140
sko100b	153890	155202	155666	156198	154688	154846	155056
sko100c	147862	148918	149482	149968	148484	148799	149034
sko100d	149576	151232	151422	151642	150676	150826	151002
sko100e	149150	150872	151189	151374	150118	150400	150638
sko100f	149036	150508	150899	151074	150118	150277	150410
sko42	15812	15934	16015	16126	15888	15902	15932
sko49	23386	23578	23724	23856	23458	23505	23558
sko56	34458	34904	35001	35112	34636	34692	34730
sko64	48498	49048	49114	49266	48790	48852	48944
sko72	60402	66906	67242	67438	66696	66814	66922
sko81	82277	91728	91944	92160	91548	91672	91746
sko90	115534	116954	117152	117306	116106	116314	116508
tho150	8134056	8222654	8247390	8257334	8193460	8202570	8208702
tho40	240516	243512	244750	246722	242000	242459	242948
wil100	273044	274630	274752	274918	273964	274106	274262

Table III. Summary of runs: best known solution (bks), minimum, average, and maximum GRASP solutions for 32 and 2048 iteration runs (problems of size n, $40 \leq n \leq 150$)

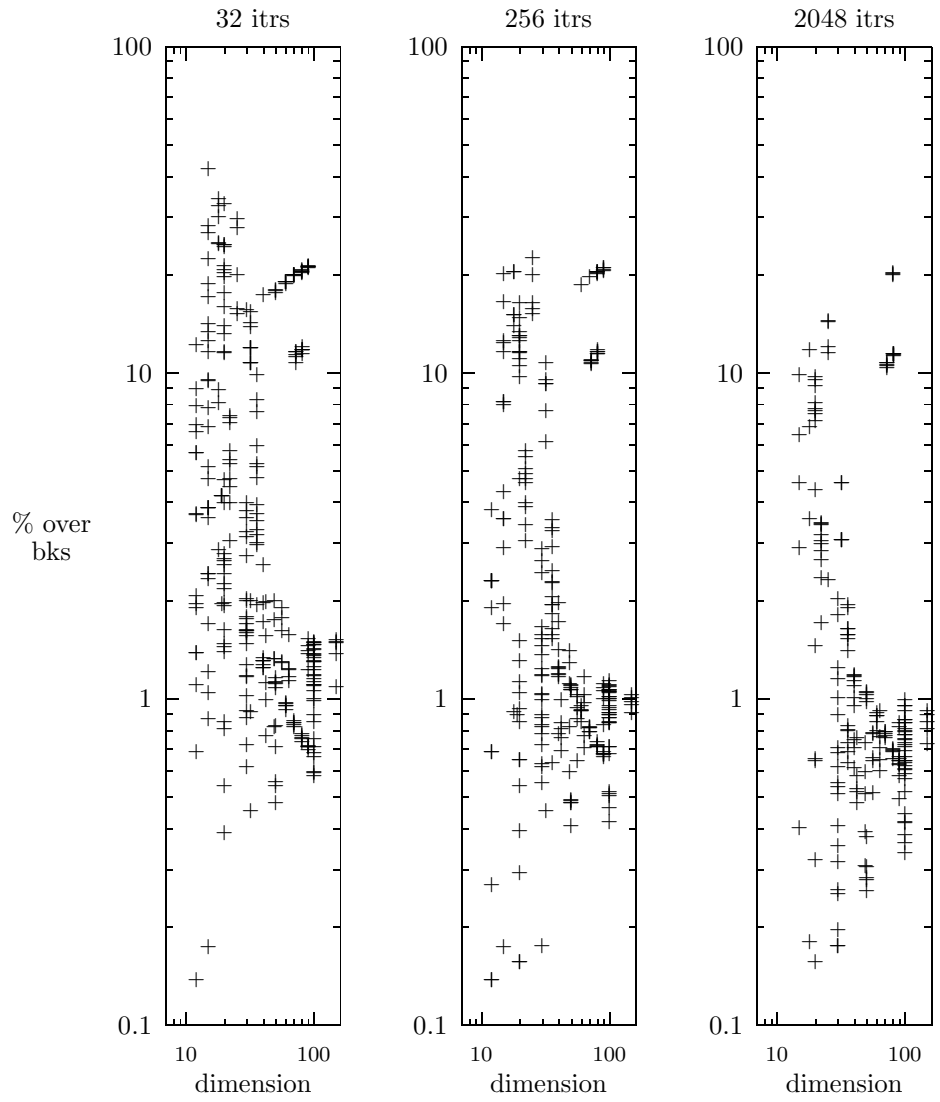

Fig. 3. Percentage over best known solution (bks) as a function of problem dimension and GRASP iterations (instances with solutions less than 0.1% of bks are not shown)

between .5% and 1% of the bks, respectively. The complete set of data, showing how solution quality is a function of GRASP iterations, is plotted in Figure 5. In that figure, the average percentage over the best known solution, as well as the 95% confidence interval are indicated by the solid lines.

Running times, for all 32, 128, 512, and 2048 iteration runs, are shown in Figure 6.

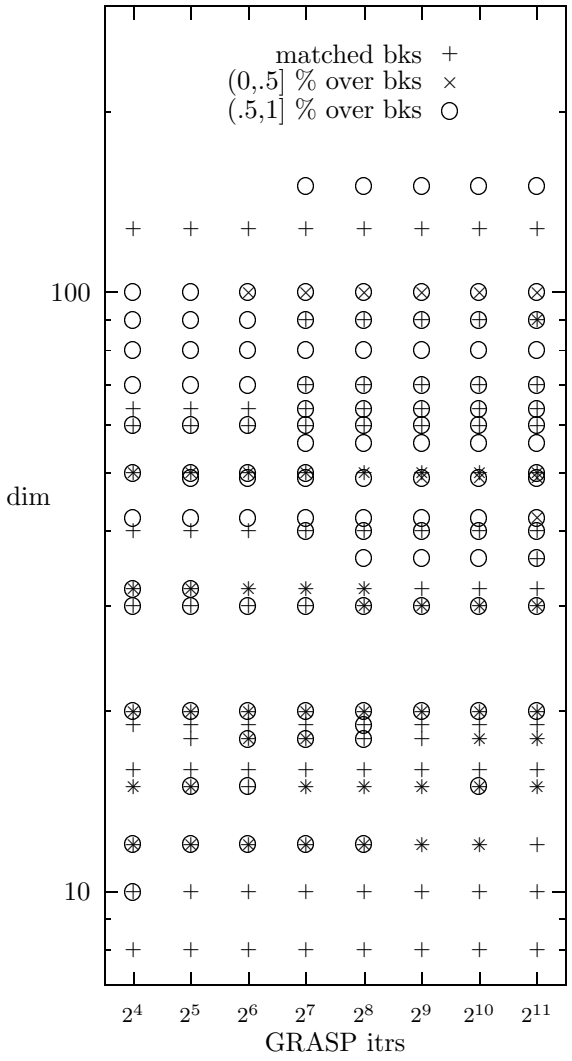

Fig. 4. Percentage over best known solution (bks) as a function of problem dimension and GRASP iterations (instances with solutions less than or equal to 1% of bks)

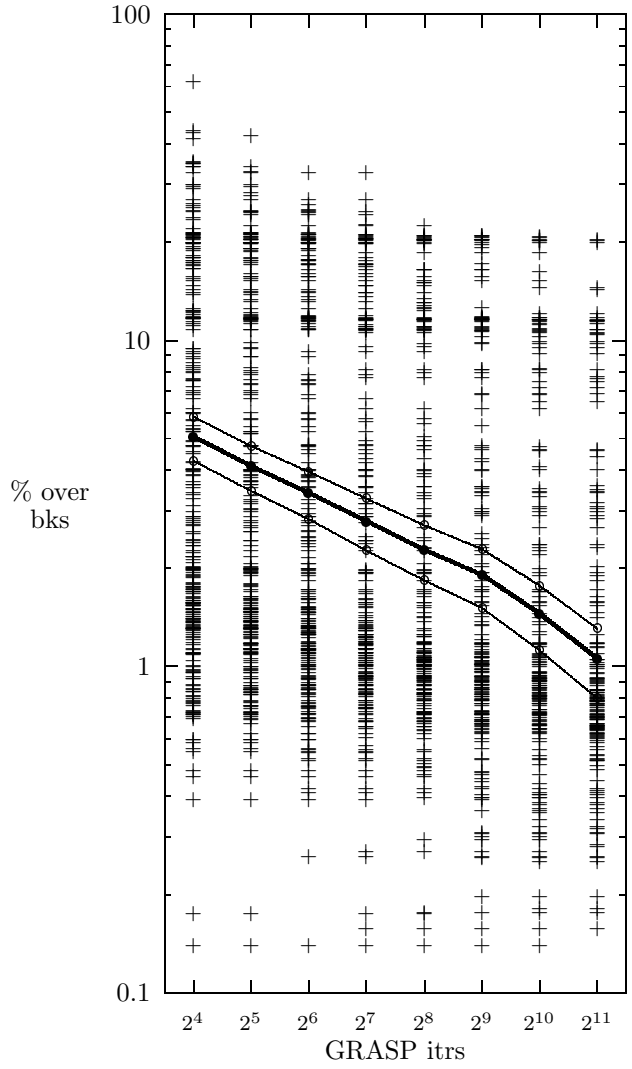

Fig. 5. Percentage over best known solution (bks) as a function of GRASP iterations (solutions within 0.1% of bks not shown)

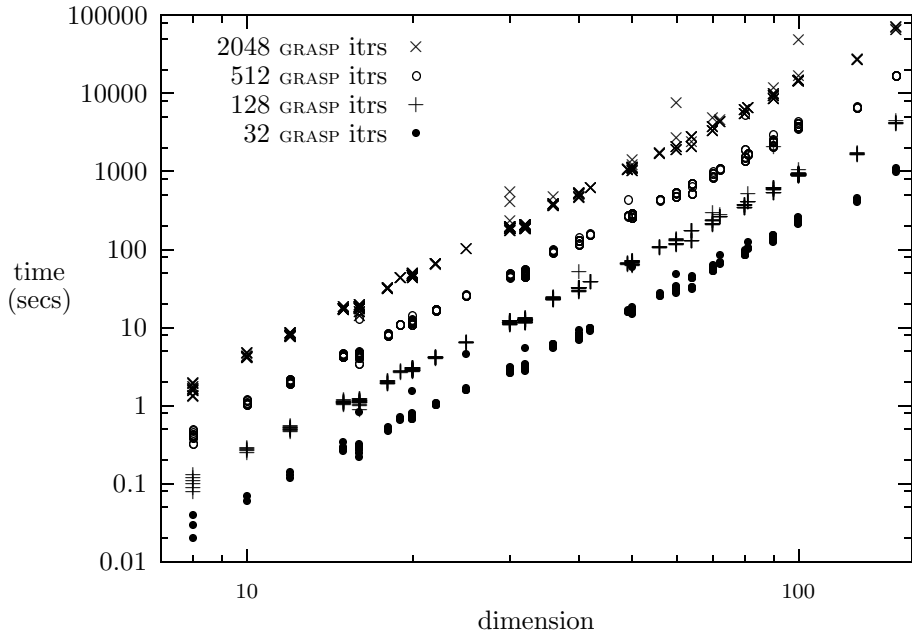

Fig. 6. Running time as a function of problem dimension and number of GRASP iterations

5. CONCLUDING REMARKS

In this paper, a set of FORTRAN subroutines that implement a GRASP for finding approximate solutions to dense symmetric assignment problems is described. The code is shown to efficiently produce good quality solutions for the test set QAPLIB. It consistently outperforms Algorithm 608 [9], a code published in *ACM Transactions of Mathematical Software* for the approximate solution of the quadratic assignment problem. It can be used on a stand-alone basis as a heuristic, as well as part of a branch and bound algorithm to quickly produce good quality initial upper bounds.

The subroutines can be easily adapted for use in an N-processor parallel computer, by making N parallel calls to the subroutine `gqapd`, each with a copy of the subroutine's variables and arrays, and a different random number seed. The best permutation found by each of the N calls is returned to the main program, which picks the overall best as the GRASP solution.

Several improvements to the code are envisioned:

– handling sparse flow and distance matrices,
– more sophisticated local search, such as 3-exchange,
– dropping the symmetry requirement,
– handling QAPs with a linear term.

REFERENCES

1. Burkard, R., Karisch, S., and Rendl, F. QAPLIB – a quadratic assignment problem library. *European Journal of Operations Research 55* (1991), 115–119. Updated version – Feb. 1993.
2. Dicky, J., and Hopkins, J. Campus building arrangement using TOPAZ. *Transportation Research 6* (1972), 59–68.
3. Elshafei, A. Hospital layout as a quadratic assignment problem. *Operations Research Quarterly 28* (1977), 167–179.
4. Feo, T., and Resende, M. Greedy randomized adaptive search procedures. Tech. rep., AT&T Bell Laboratories, Murray Hill, NJ 07974-2070, February 1994. To appear in *Journal of Global Optimization*.
5. Koopmans, T., and Beckmann, M. Assignment problems and the location of economic activities. *Econometrica 25* (1957), 53–76.
6. Li, Y., Pardalos, P., and Resende, M. 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. Pardalos, P., Rendl, F., and Wolkowicz, H. The quadratic assignment problem: A survey and recent developments. 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. 1–42.
8. Schrage, L. A more portable Fortran random number generator. *ACM Transactions on Mathematical Software 5* (1979), 132–138.
9. West, D. Algorithm 608: Approximate solution of the quadratic assignment problem. *ACM Transactions on Mathematical Software 9* (1983), 461–466.