MatQapNB User Guide: A branch-and-bound program for **QAP**s in **Mat**lab with the **N**ewton-**B**racketing method

Koichi Fujii[*], Naoki Ito[†], Sunyoung Kim[‡], Masakazu Kojima[§], Hans D. Mittelmann[¶], Yuji Shinano[‖], Kim-Chuan Toh[**]

June 9, 2021

Abstract

MatQapNB is a MATLAB toolbox that implements a parallel branch-and-bound method using NewtBracket (the Newton-bracketing method [4]) for its lower bounding procedure. It can solve small to medium scale Quadratic Assignment Problem (QAP) instances with dimension up to 30. MatQapNB was used in the numerical experiments on QAPs in the recent article "Solving challenging scale QAPs" [3] by Fujii, Ito, Kim, Kojima, Shinano and Toh, an intermediate progress report on the authors' joint project for solve large-scale QAP instances in QAPLIB whose exact optimal values are unknown [2]. For large scale QAPs, the authors have been developing a C++ version of the method, **QapNB**, which works on the Ubiquity Generator (UG) framework utilizing more than 100,000 cores, based on MatQapNB. We refer to the article [3] for theoretical and technical details on the parallel branch-and-method implemented in MatQapNB. This user guide describes the usage of MatQapNB. MatQapNB is included in the software package NewtBracket-v2.0 as an application, which can be obtained from
https://sites.google.com/site/masakazukojima1/softwares-developed/newtbracket

1 Quadratic assignment problems (QAP)s

MatQapNB is for solving the quadratic assignment problem (QAP). Let $N = \{1, \ldots, n\}$ represent a set of locations and also a set of facilities. Given $n \times n$ **symmetric matrices** $\boldsymbol{A} = [a_{ik}]$ and $\boldsymbol{B} = [b_{j\ell}]$, the QAP is stated as

$$\min_{\pi} \sum_{i \in N} \sum_{k \in N} a_{ik} b_{\pi(i),\pi(k)} \tag{1}$$

[*]NTT DATA Mathematical Systems Inc., Tokyo
[†]Fast Retailing Co., Ltd., Uniqlo City Tokyo
[‡]Department of Mathematics, Ewha W. University, Seoul
[§]Department of Industrial and Systems Engineering, Chuo University, Tokyo
[¶]School of Mathematical and Statistical Sciences, Arizona State University, Tempe, Arizona
[‖]Department of Applied Algorithmic Intelligence Methods (A^{2}IM), Zuse Institute Berlin, Berlin
[**]Department of Mathematics, and Institute of Operations Research and Analytics, National University of Singapore, Singapore

where a_{ik} denotes the flow between facilities $i \in N$ and $k \in K$, $b_{j\ell}$ the distance between locations $j \in N$ and $\ell \in N$, and $(\pi(1), \ldots, \pi(n))$ a permutation of $1, \ldots, n$ such that $\pi(i) = j$ if facility $i \in N$ is assigned to location $j \in N$. All elements of $\boldsymbol{A}$ and $\boldsymbol{B}$ are assumed to be **nonnegative integers**.

To solve a QAP instance, the user needs to prepare a data file *.dat of the following form:

n
$\boldsymbol{A}$
$\boldsymbol{B}$

For example, the data file example.dat of a QAP instance with

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 3 & 0 \end{pmatrix}, \ \boldsymbol{B} = \begin{pmatrix} 0 & 4 & 5 \\ 4 & 0 & 6 \\ 5 & 6 & 0 \end{pmatrix}$$

is given by

```
3

0 1 2
1 0 3
2 3 0

0 4 5
4 0 6
5 6 0
```

To solve example.dat, issue a command as

```
>> [solQAP,infoQAP] = MatQapNB('example');
```

We note that MatQapNB computes an optimal solution by enumerating all feasible solutions if $n \leq 9$, so the branch-and-bound procedure is not applied to the above case.

2 QAP instances from QAPLIB [1, 2, 5]

The data files of many QAP instances from [1, 5] (see also [2] for the latest updates on upper and lower bounds of large scale QAP instances) are stored in the directory NewtBracket-v.2/instances/QAP/qapdata, which includes

- 'bur26a', 'bur26b', 'bur26c', 'bur26d', 'bur26e', 'bur26f', 'bur26g', 'bur26h (Set paramNB.branchSW = 3 to solve by MatQapNB)
- 'chr12a', 'chr12b', 'chr12c', 'chr15a', 'chr15b', 'chr15c', 'chr18a', 'chr18b', 'chr20a', 'chr20b', 'chr20c', 'chr22a', 'chr25a'
- 'els19'
- 'had12', 'had14', 'had16', 'had18', 'had20'

- 'kra30a', 'kra30b', 'kra32'
- 'lipa20a', 'lipa20b','lipa30a', 'lipa30b'
- 'nug12', 'nug14', 'nug15', 'nug16a', 'nug16b', 'nug17', 'nug18', 'nug20', 'nug21', 'nug22', 'nug24', 'nug25', 'nug27',' nug28'
- 'scr12', 'scr15', 'scr20'
- 'tai10a', 'tai10b', 'tai12a', 'tai12b', 'tai15a', 'tai15b', 'tai17a', 'tai20a', 'tai20b', 'tai25b', ' tai30b'
- 'tai25a' (Set paramNB.branchSW = 3 to solve by MatQapNB)

The command to solve an instance listed above, for example, nug12.dat, is:

```
>> [solQAP,infoQAP] = MatQapNB('nug12');
```

Then the following is displayed on the screen as MatQapNB terminates:

```
# nug12 terminated: No. of ActiveNodes = 0
# QAPname=nug12, paramNB:
# fileTail=BB03-Jun-2021 randSeed=64700 paraSW=1 printSW=-1
# saveFileSW=-1 optVal=578 bestObjVal=578 qapDim0=12
# compSW=0# lambda=+1.0000e+05 intValSW=1 LB=0
# UB=5.905800000e+02 rho=13 branchSW=2 warmStartSW=0#
eTime: LowerBound=1.02e+01, UpperBound=7.05e-01, Branching=1.32e+01,
Total=1.85e+01 in para
# The best obj. val. computed = 578
# The best feas. sol. computed =
  (3 9 7 12 1 11 8 4 2 10 6 5 )
# The best LB computed = 578
# LB sequence: qapDim-LB
# 12-514 11-578
# dim-'No. of nodes(subQAPs(DNNs) generated)' : 12-1 11-12
# dim-'No. of nodes(subQAPs(DNNs) solved)'    : 12-1 11-12
# Total no. of nodes(subQAPs(DNNs) generated) = 13
# Total no. of nodes(subQAPs(DNNs) solved)    = 13
# The average time to solve one DNN for LB = 7.86e-01
%      &  The best  &    & No. of  & \multicolumn{4}{|c|}{Time(sec)}\\
% QAP  &  known UB  & Br & nodes   & Total in para. &    LB    &    UB    & Br \\
nug12  &        578 &  2 &     13  &    1.85e1      & 1.02e1   & 7.05e-1  & 1.32e1 \\
```

Remark 2.1. The optimal value of every QAP instance listed above from QAPLIB [1, 5] is known. MatQapNB sets the initial incumbent optimal value, which is given by the parameter paramNB.bestObjVal, to its optimal value + 1 by default. For other QAP instances, the user needs to specify the initial incumbent optimal value with the parameter paramNB.bestObjVal, or the default value, paramNB.bestObjVal = the mean of objective values over all feasible solutions, can be used. See Section 6.

3 Getting started with some examples

The simplest way is to execute the script startQAP.mat with specifying QAPname as follows:

```
% startMatQaPNB.m
clear all;
paramNB = [];
QAPname = 'nug12';
[solQAP,infoQAP] = MatQapNB(QAPname,paramNB);
```

We present three examples, Examples 3.1, 4.1 and 5.1, to describe some basic and useful parameters to control the execution and the output of MatQapNB,

Example 3.1.

```
% QapExample31.m
%
clear all
%
% Major parameters; See Section 6 for more details.
%
% paramNB.printSW = -1; print the final results only on the screen --- default
% paramNB.printSW = 0 ; print the final results on the screen and write them
%        on the file printFileName = [QAPname,paramNB.fileTail,'.txt'];
% paramNB.printSW = 1; print the intermediate and final results on the screen
%        and write them on the file
% printFileName = [QAPname,paramNB.fileTail,'.txt'];
% paramNB.printSW = 2; write the intermediate and final results on the file
%        diaryFileName = [QAPname,'Diary',paramNB.fileTail,'.txt'];
% Here the default value of paramNB.fileTail is 'BB\date' if paramNB.targetLB is
%        not specified or paramNB.targetLB = [], and 'LB\date'
%        if paramNB.targetLB is specified.
% paramNB.branchSW = 1; apply the branching rule M;
% paramNB.branchSW = 2; apply the branching rule P; --- default
% paramNB.branchSW = 3; apply the branching rule D;
% paramNB.branchSW = 4; apply the branching rule G;
% See Section 5.1, 5.2 and 5.3 of
% "Solving Challenging Large QAPs", ZIB-REP-21-02
% for the branching rules M, P and D, respectively. The branching rule G
% was not mentioned there but utilizes the Gilmore-Lawler bound
% of Q$((F,f),(L,\ell))$
% for $\varphi(f,\ell)$ in the branching rules described in Section 5.
%
% paramNB.saveFileSW = 0; ---> output no information to any mat file --- default
% paramNB.saveFileSW = 1;
% ---> output the final values on the MATLAB variables:
%       QAPname,paramNB,iteration,bestObjVal,bestSolComputed,node,
```

```
% infoAllIt,activeNodes
%       to a mat file [instance,'LastIt',paramNB.fileTail,'.mat'];
% paramNB.saveFileSW =  2;
% ---> output the intermediate and final values on the MATLAB variables
%      to mat files.
% The mat file saved can be used for restart MatQapNB. See Section 5.

QAPname = 'tai15a';
paramNB.printSW = 1;
% ---> The final results are written on both screen and
%        the file ['tai15a','BB\date','.txt']
paramNB.branchSW = 1;
paramNB.saveFileSW = 1;
% ---> output the final values on the MATLAB variables:
%       QAPname,paramNB,iteration,bestObjVal,solQAP,node,infoQAP,activeNodes
%       to a mat file ['tai15a''LastIt','BB\date','.mat'];
[solQAP,infoQAP] = MatQapNB(QAPname,paramNB);
```

Numerical results on some of QAP instances from QAPLIB [1, 5] are presented in Tables 1, 2, 3 and 4 of of [3].

4 Estimating the number of nodes to generate for the branch-and-method

This function is to determine how large QAP instances the branch-and-bound method implemented in MatQapNB could solve, when it is implemented in C++ on a more powerful parallel branch-and-bound framework, the Ubiquity Generator (UG) framework utilizing more than 100,000 cores. The method for this function is presented in Section 7.2 of [3].

In short, the number of the nodes generated by the branch-and-bound method is estimated by randomly constructing a subtree of the enumeration tree. To execute this function, we need to specify paramNB.sampleSize which determines the maximum number of nodes sampled at each depth of the subtree constructed.

Example 4.1. We consider nug20 as an example and set paramNB.sampleSiz = 40.

```
% QapExample41.m
clear all
QAPname = 'nug20';
paramNB.printSW = 2;
paramNB.branchSW = 2;
paramNB.sampleSize = 40;
paramNB.saveFileSW = 1;
paramNB.fileTail = 'Sampling';
[solQAP,infoQAP] = MatQapNB(QAPname,paramNB);
```

As a result, we have:

```
##### saved nug20LastSampling.mat:
 QAPname,paramNB,iteration,bestObjVal,solQAP,oneCNodeIdx,node,
 infoEachIt,infoQAP,activeNodes #####
 . . . . . .
# eTime: LowerBound=1.14e+03, UpperBound=1.53e+02, Branching=4.51e+01,
Total=4.33e+02 in para
 . . . . . .
The average time to solve one DNN for LB = 8.03e+00
%% Estimation of the number of nodes of the enumeration tree \\
% s_r = the number of selected nodes <= paramNB.sampleSize \\
% t_r = the number of nodes remained active among the selected nodes\\
% c_r = the total number of child nodes of the nodes remained active\\
% hatm_{r+1}   = hatm_r   * (n-r) * (t_r/s_r) \\
% tildem_{r+1} = tildem_r * c_r   * (1/s_r) \\
% hatm is explaind Section 7.2 of ZIPreport-21-02,\\
% tildem is more reasonable; in general c_r <= (n-r) * t_r; \\
% but, in many cases, c_r = (n-r) * t_r and tildem = hat m. \\
   r  & hatm_r & (n-r) & t_r/s_r &   1/s_r &    c_r & tilde{m}_r \\
   0  &        1 &    20 & 1.00000 & 1.00000 &     20 &      1 \\
   1  &       20 &    19 & 0.50000 & 0.05000 &    190 &     20 \\
   2  &      190 &    18 & 0.05000 & 0.02500 &     36 &    190 \\
   3  &      171 &    17 & 0.00000 & 0.02778 &      0 &    171 \\
Total &      382 &       &         &         &        &    382 \\
% Estimate of the total number of nodes : hatm = 382 or tildem = 382 \\
% Summary of estimation: \\
 . . . . . .
```

In addition, we obtain the text file nug20Sampling.txt and the mat file nug20LastSampling.mat which store the values of variables for restarting MatQapNB from the mat file to complete the branch-and-bound method for solving the instance nug20. Restarting MatQapNB from nug20LastSampling.mat will be described in the next section.

Estimating the number of nodes for larger scale instances is given in Table 2 of Section 7. See also Table 5 of [3].

5 Restarting MatQapNB

If the values of some MATLAB variables are stored at the end of while loop of MatQapNB, then MatQapNB can be restarted. Setting paramNB.saveFileSW = 1 (or paramNB.saveFileSW = 2) is to store the values of some MATLAB variables in a *.mat file. The mat file name is given by
[QAPname,'It',num2str(iteration),paramNB.fileTail,'.mat'],
if the variables are stored at the end of each iteration with paramNB.saveSW = 2, or
[QAPname,'Last',paramNB.fileTail,'.mat']
if the variables are stored just before the termination with paramNB.saveSW = 1 or 2. The

latter case occurs when MatQapNB is used to generate a subset of the nodes for estimating the number of the nodes on the enumeration tree, as mentioned in Section 4. If restartFileName is the name of the *.mat file, we can restart MatQapNB from the *.mat file by issuing "MatQapNB(restartFile,paramNB)". Here, the paramNB stored in the *.mat file is overwritten by the input paramNB; specifically paramNB.sampleSize = [] needs to be specified to complete the branch-and-bound method from the previous estimation.

Example 5.1. This example illustrates how to restart MatQapNB from 'nug20LastSampling.mat' which was generated in Example 4.1.

```
% QapExample51.m
clear all
restartFileName = 'nug20LastSampling.mat';
paramNB.printSW = 1;
paramNB.sampleSize = [];
paramNB.saveFileSW = 0;
paramNB.fileTail = 'Restart';
[solQAP,infoQAP] = MatQapNB(restartFileName,paramNB);
```

As a result, we obtain:

```
# nug20 terminated: No. of ActiveNodes = 0
# QAPname=nug20, paramNB:
# printSW=1 saveFileSW=1 fileTail=Restart paraSW=1
# optVal=2570 bestObjVal=2570 qapDim0=20 compSW=0
## lambda=+1.0000e+05 intValSW=1 LB=0 UB=2.622420000e+03
# rho=21 branchSW=2 randSeed=64700 warmStartSW=0# eTime:
LowerBound=1.71e+03, UpperBound=9.72e+01, Branching=1.24e+01,
Total=1.51e+02 in para
# The best obj. val. computed = 2570
# The best feas. sol. computed =
  (6 1 7 5 17 13 8 20 15 19 16 11 12 2 4 9 3 10 14 18 )
# The best LB computed = 2570
# LB sequence: qapDim-LB
# 20-2057 19-2237 18-2560 17-2570 18-2353 17-2570
# dim-'No. of nodes(subQAPs(DNNs) generated)' : 20-1 19-20 18-190 17-54
# dim-'No. of nodes(subQAPs(DNNs) solved)'    : 20-1 19-20 18-190 17-54
# Total no. of nodes(subQAPs(DNNs) generated) = 265
# Total no. of nodes(subQAPs(DNNs) solved)    = 265
# The average time to solve one DNN for LB = 6.47e+00
```

We recall that the estimation of the total number of nodes given in Example 4.1 was 382.

6 Input and output of MatQapNB: instance, paramNB, solQAP, and infoQAP

Input instance

- instance = a file containing the data of a QAP instance, the dimension n, the $n \times n$ flow matrix $\boldsymbol{A}$ and the $n \times n$ distance matrix $\boldsymbol{B}$, where both $\boldsymbol{A}$ and $\boldsymbol{B}$ are assumed to be symmetric, nonnegative and integer. The file extension should be '.dat', but the input file name does not include the extension, for example, if the file is QAPexample.dat, then instance = 'QAPexample'. See Section 1 and also the directory NewtBracket-v.2/instances/QAP/qapdata for the data format.
- instance = '*.mat', a mat file that contains the values of some MATLAB variables stored in the previous execution of MatQapNB before it terminated and/or the values of variables to further compute for an optimal solution. We can restart MatQapNB from the mat file. See Examples 4.1 and 5.1

Table 1: **Input paramNB**. Default values† are defined in defaultParamsBrQAP. ‡ : paramNB.paraSW should be set to 0 to interrupt the program for debugging.

Field of paramNB	Default values†	Possible values and their meaning
paraSW‡	1	1 for parallel computation, 0 for serial computation.
printSW	-1	-1, 0, 1, 2. See Example 3.1.
fileTail	" (Empty string)	Any string. See Example 3.1.
saveFileSW	0	0, 1, 2. See Examples 3.1 and 4.1.
noOfThreads	[]	the number of threads in parallel computation. See script files in the directory BandBestimation.
sampleSize	[] (Emptyset)	[], any positive integer. See Example 4.1.
optVal	getOptVal(Instance)	The known best UB for the QAP instances from QAPLIB [1].
	[] (Emptyset)	for other instances.
bestObjVal		The initial incumbent objective value.
	optVal+1	If paramNB.optVal $\neq$ []. Else
	the mean of objective values over all feasible solution. See Remark 2.2	
branchSW	2	1, 2, 3, 4. See Example 3.1 and Section 5 of [3].
maxiterMatQapNB	[]	The max. number of iterations of MatQapNB.
maxNoOfNodes	[]	The max. number of nodes processed.
targetLB	[]	The target lower bound. See Section 7.
intValSW	1	1 if the unknown optimal value is integer, and 0 otherwise.

Output solQAP

- solQAP.optVal = the optimal value of the QAP instance solved.
- solQAP.perm = an optimal solution π of the QAP instance solved in terms of a permutation. See (1).

Output infoQAP

- infoQAP.noOfNodes = an n-dimensional row vector $[t_1, \ldots .t_n]$ whose element t_k denotes the number of nodes on the enumeration tree corresponding to k-dimensional sub QAPs, where n denotes the dimension of the root node QAP, the QAP instance solved.
- infoQAP.totalNoOfNodes = the total number of nodes on the enumeration tree.
- infoQAP.timeTotal = the total execution time for solving the given QAP instance.
- infoQAP.timeForLowerBound = the accumulation of the time for computing lower bounds in parallel.
- infoQAP.timeForUppeBound = the accumulation of the time for computing lower bounds in parallel.
- infoQAP.timeForBranch = the total execution time for branching process.
- infoQAP.LBsequence = Each pair of elements in the ith row of this $k \times 2$ matrix denotes the dim. q_i of the subQAP in the enumeration tree and the lower bound obtained there, where the iteration has stopped at subQAPs with dim. q_k.

7 Lower bound computation

As MatQapNB can be used for computing lower bounds (LBs) of a given QAP instances without solving it, this function is designed to update the known LB of a large scale, hard QAP instance. Those large QAP instances with their best upper and lower bounds are listed in [2]. To illustrate the usage, we apply the function to a small QAP instance below.

A simple way is to execute the following script with the information on QAP names and LBs: QAPname = 'instance QAP name' and paramNB.targetLB = 'the LB to be attained'.

```
% A simple computation of LB = 3530 of 'nug22' --->
clear all
paramNB = [];
QAPname = 'nug22';
paramNB.targetLB = 3530;
paramNB.fileTail = 'LB3530';
[solQAP,infoQAP] = MatQapNB(QAPname,paramNB);
% <--- A simple computation of LB = 3530 of 'nug22'
```

We then obtain

```
# nug22 terminated: No. of ActiveNodes = 0
# QAPname=nug22, paramNB:
# targetLB=3530 fileTail=LB3530 randSeed=64700 paraSW=1
# printSW=-1 saveFileSW=-1 optVal=3596 bestObjVal=3596
# qapDim0=22 compSW=0# lambda=+1.0000e+05 intValSW=1
# LB=0 UB=3.600600000e+03 rho=23 branchSW=2
# warmStartSW=0
# eTime: LowerBound=2.07e+02, UpperBound=0.00e+00, Branching=1.18e+01,
Total=1.04e+02 in para
```

```
# The best LB computed = 3530
# LB sequence: qapDim-LB
# 22-3518 21-3530
# dim-'No. of nodes(subQAPs(DNNs) generated)' : 22-1 21-22
# dim-'No. of nodes(subQAPs(DNNs) solved)'    : 22-1 21-22
# Total no. of nodes(subQAPs(DNNs) generated) = 23
# Total no. of nodes(subQAPs(DNNs) solved)    = 23
# The average time to solve one DNN for LB = 9.02e+00
```

We can specify the parameter paramNB.sampleSize to estimate the amount of computational work in terms of the number of nodes or the number of DNN subproblems to attain a given paramNB.targetLB. This technique has been discussed for MatQapNB to compute the exact optimal value of a given QAP instance. As an example, we consider the script:

```
% Estimation of computational work to attain paramNB.targetLB by sampling --->
clear all
paramNB = [];
QAPname = 'nug22';
paramNB.targetLB = 3580;
paramNB.sampleSize = 50;
paramNB.saveFileSW = 1;
paramNB.fileTail = 'Sampling50LB3580'; % option
[solQAP,infoQAP] = MatQapNB(QAPname,paramNB);
% <--- Estimation of computational work to attain paramNB.targetLB by sampling
```

As a result, we get

```
##### saved nug22LastSampling50LB3580.mat:
 QAPname,paramNB,iteration,bestObjVal,solQAP,oneCNodeIdx,node,infoEachIt,
 infoQAP,activeNodes #####
# nug22 : sampling terminated. Sample size = 50
# QAPname=nug22, paramNB:
# targetLB=3580 sampleSize=50 fileTail=Sampling50LB3580 randSeed=9756
# paraSW=1 printSW=-1 saveFileSW=-1 optVal=3596
# bestObjVal=3596 qapDim0=22 compSW=0# lambda=+1.0000e+05
# intValSW=1 LB=0 UB=3.651600000e+03 rho=23
# branchSW=2 warmStartSW=0
# eTime: LowerBound=9.73e+02, UpperBound=0.00e+00, Branching=1.29e+01,
Total=2.80e+02 in para
# The best LB computed = 3580
# LB sequence: qapDim-LB
# 22-3108 21-3330 20-3564 19-3580
# dim-'No. of nodes(subQAPs(DNNs) generated)' : 22-1 21-22 20-84 19-20
# dim-'No. of nodes(subQAPs(DNNs) solved)'    : 22-1 21-22 20-50 19-20
# Total no. of nodes(subQAPs(DNNs) generated) = 127
# Total no. of nodes(subQAPs(DNNs) solved)    = 93
# The average time to solve one DNN for LB = 1.05e+01
% Estimation of the number of nodes of the enumeration tree \\
```

```
% r   = the depth of the enumeration tree \\
% s_r = the number of selected nodes <= paramNB.sampleSize \\
% t_r = the number of nodes remained active among the selected nodes\\
% c_r = the total number of child nodes of the nodes remained active\\
% hatm_{r+1}   = hatm_r   * (n-r) * (t_r/s_r) \\
% tildem_{r+1} = tildem_r * c_r   * (1/s_r) \\
% hatm is explaind in Section 7.2 of ZIPreport-21-02,\\
% tildem is more reasonable; in general c_r <= (n-r) * t_r; \\
% but, in many cases, c_r = (n-r) * t_r and tildem = hatm. \\
      &          & qapDim &      &      & t_r/s_r &   1/s_r &    c_r & tilde{m}_r \\
   r  &  hatm_r  & (n-r)  &  s_r &  t_r & t_r/s_r &   1/s_r &    c_r & tilde{m}_r \\
   0  &        1 &     22 &    1 &    1 & 1.00000 & 1.00000 &     22 &      1 \\
   1  &       22 &     21 &   22 &    4 & 0.18182 & 0.04545 &     84 &     22 \\
   2  &       84 &     20 &   50 &    1 & 0.02000 & 0.02000 &     20 &     84 \\
   3  &       34 &     19 &   20 &    0 & 0.00000 & 0.05000 &      0 &     34 \\
Total &      141 &        &         &         &        &    141 \\
% The estimate number of DNN to be solved = sum tildem(initDepth:end) = 141. \\
% Summary of estimation: \\
%      &             &         Current       &  Depth     & Sample &      Target
&  DNNs  to   & Average time\\
% QAP  &    OptVal   &         LB(Gap)       & init.(end) & size   &      LB(Gap)
&  be solved  & for one DNN \\
nug22 &       3596  &           (    ) &       0(3) &   50  &       3580(4.449e-03)
&         141 & 1.046e+01 \\
```

Thus, we see that the computation of LB = 3580 is expected to require solving 141 DNN subproblems.

We can restart MatQapNB from nug22LastSampling50LB3580.mat by executing the following script:

```
% Restart from 'nug22LastSampling50LB3580.mat'  for paramNB.targetLB = 3580 --->
clear all
paramNB = [];
QAPname = 'nug22LastSampling50LB3580.mat';
% paramNB.targetLB = 3580;
paramNB.sampleSize = []; % Reset this parameter;
paramNB.saveFileSW = 0; % Reset this parameter;
paramNB.fileTail = []; %or 'LBrestart3580';
[solQAP,infoQAP] = MatQapNB(QAPname,paramNB);
% <--- Restart from 'nug22LastSampling50LB3580.mat' for paramNB.targetLB = 3580
```

Table 2 shows numerical results on the estimation of the number of DNN subproblems to attain the gap between the known UB and a given target LB for large scale unsolved QAP instances. We note that Current LB (Gap) is from QAPLIB [2]. The script files are provided in the directory NewtBracket-v.2/applications/MatQapNB/BandBestimation for this numerical experiment.

Table 2: Estimation of the number of DNN subproblems to be solved (= the number of nodes generated in the branch-and-bound enumeration tree) for halving the gap between the known UB and a target LB. Depth: the depth of the enumerations tree where DNN subproblems were solved for the target LB. Br.: the branching procedure (M: Mean of sub QAPs, P: Primal DNN subproblems, see Section 5 of [3]).

QAP	e OptVal	Current LB(Gap)	Depth, Br.	Sample size	Target LB(Gap)	DNNs to be solved	Average time for one DNN
tai35a	2,422,002	2,216,627(8.48e-2)	0→4, P	192	2,325,122(4.00e-2)	9.90e5	1.20e3
tho40	240,516	226,490(5.83e-2)	0→8, P	192	240,516(0.00e0)	8.84e7	7.96e2
tho40	240516	226,490(5.83e-02)	0→8, M	192	240,516(0.00e0)	9.83e8	8.79e2
tai40a	3,139,370	2,843,274(9.43e-2)	0→5, P	192	2,999,061(4.47e-2)	3.49e6	1.59e3
sko49	23,386	22,650(3.15e-2)	0→9, P	192	23,386(0.00e0)	7.23e8	4.76e3
sko49	23,386	22,650(3.15e-2)	0→8, M	192	23,386(0.00e0)	3.83e8	4.87e3
tai50a	4,938,796	4,390,920(1.11e-1)	0→4, P	192	4,568,387(7.50e-2)	5.13e6	6.04e3
tai50b	458,821,517	431,090,700(6.044e-2)	0→6, M	192	458,821,517(0.00e0)	4.60e7	5.94e3
wil50	48,816	48,121(1.424e-2)	0→9, P	192	48,816(0.00e0)	1.22e8	6.33e3
wil50	48,816	48,121(1.424e-2)	0→8, M	192	48,816(0.00e0)	4.07e7	4.68e3
sko56	34,458	33,385(3.11e-2)	0→7, P	192	34,390(2.0e-3)	7.03e8	1.48e4
tai60a	7205962	6325978(1.221e-01)	0→4, P	144	6557426(9.000e-02)	1.17e7	1.25e4
tai60b	608,215,054	592,371,800(2.61e-2)	0→11, P	144	608,215,054(0.00e0)	9.63e8	8.95e3
tai60b	608,215,054	592,371,800(2.61e-2)	0→8, M	144	608,215,054(0.00e0)	7.29e7	1.94e4
sko64	48,498	47,017(3.054e-2)	0→5, P	96	48,014(9.980e-3)	7.16e7	2.27e4
sko72	66256	64455(2.718e-2)	0→7, P	96	65,594(9.992e-3)	2.57e8	3.76e4
tai80a	13,499,184	11,657,010(1.365e-1)	0→9, P	96	12,419,250(8.000e-2)	8.10e16	8.09e4
tai80b	818,415,043	786,298,800(3.92e-2)	0→8, P	96	810,230,893(1.000e-2)	1.31e9	6.57e4

Acknowledgments

The work of M. Kojima was supported by Grant-in-Aid for Scientific Research (A) 19H00808. The work of H. D. Mittleman was supported in part by the Air Force Office of Scientific Research under grant FA9550-19-1-0070. The work of Y. Shinano was supported by the Research Campus MODAL funded by the German Federal Ministry of Education and Research (fund number 05M14ZAM).

References

[1] QAPLIB – A Quadratic Assignment Problem Library, Computtonal Optimization Research at Lehigh. https://coral.ise.lehigh.edu/data-sets/qaplib/, August 2011.

[2] M. Anjos. "QAPLIB is a a Quadratic Assignment Problem Library" in Miguel Anjos' Homepage. https://www.miguelanjos.com/qaplib.

[3] K. Fujii, N. Itoh, N. Kim, M. Kojima, Y. Shinano, and K. C. Toh. Solving challenging scale QAPs. Technical Report ZIB-Report-21-02, Zuse Institute Berlin, 14195 Berlin, Germany, January 2021.

[4] S. Kim, M. Kojima, and K.C. Toh. User manual of newtbracket: "A Newton-Bracketing method for a simple conic optimization problem" with aopplications to QOPs in binary variables. https://sites.google.com/site/masakazukojima1/softwares-developed/newtbracket, November 2020.

[5] S. Karisch R E. Burkard and F. Rendl. QAPLIB - A Quadratic Assignment Problem Library. *Journal of Global Optimization*, 10(1):391–343, 1997.