ST444 GROUP PROJECT

A parallel tabu search algorithm for solving quadratic assignment problems

Xuhui Li

February 2, 2021

1 Introduction

The tabu search algorithm (TS) is a heuristic algorithm that is used to determine/approximate global optimal solutions for a variety of optimisation problems. Due to its general applicability to many optimisation problems, the TS has been labelled as 'metaheuristic' (Fiechter, 1994). A defining characteristic of TS is that it seeks to add a level of 'intelligence' to the search for a global optimal solution. The 'tabu list' keeps a finite set of previous visited solutions, which acts as a short-term memory of past solutions. Any solutions already visited are prohibited from being revisited as long as they are in the 'tabu list'. Consequently, a memory of past solutions can reduce the likelihood that the TS becomes 'stuck' at local optima solutions.

In the scope of this project, the TS is an attractive algorithm to study as it has several aspects which can be parallelised. For instance, when searching local surroundings for the best solution (not within the 'tabu list'), the TS will explore multiple options and evaluate the best one. Moreover, parallel TS can also execute two different tasks - exploration and modification of tabu list simultaneously.

Rather than study the benefits of parallelising the TS on multiple optimisation problems, we have elected to focus on studying the Quadratic Assignment Problem (QAP). This is primarily because the QAP is a generalised combinatorial optimisation problem from a well-known family of problems such as facility allocation, backboard wiring, scheduling etc. Particularly, the traveling salesmen problem is a specific/simplified type of QAP. By studying the benefits of TS parallelisation on the QAP, we will develop a general understanding of the benefits of TS parallelisation on this specific family of optimisation problems.

Based on the work of Fiechter (1994) we hypothesise that the parallelised TS will provide exponentially increasing time benefits over a non-parallelised TS as the complexity of the QAP increases (the number of facilities to be allocated increases).

The report is organised as follows. Section 2 formally describes the application of the QAP. A simple introduction of parallel computing is given in section 3. Section 4 provides a broad overview of the TS and explores the finer details of the algorithm. Furthermore, section 4 details the implementation of parallel computing in TS. A flow chart illustrating TS and the TS pseudo code are provided in section 5 and 6 respectively. Section 7 provides simulation results which are used to compare the parallel and non-parallel TS implementation. Moreover, the benefits and shortcoming of TS parallelisation will be discussed. Section 8 provides future improvement, and section 9 concludes our report.

2 Quadratic assignment problems

The quadratic assignment problem (QAP) was introduced by Koopmans and Beckmann in 1957 as a mathematical model for the location of a set of indivisible economical activities (Burkard et al., 1998). The QAP is a generalised problem which encompasses the travelling salesman problem.

Specifically, the QAP is a problem which aims to minimise the cost of assigning n facilities to n locations. The distance between each location and the 'flow' between each facility is specified. The cost function is the sum product of the distance and flow between facilities located at each location.

Mathematically, let each facility be denoted by f_i with i = {1,2, ..., n}; location denoted by l_k with k ∈ {1,2, ..., n}.

Define a flow matrix W representing the flow between each facility f_i & f_j with i,j ∈ {1,2, ..., n}:

$$W = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,n} \end{pmatrix}$$

Where $w_{i,i} = 0$ for ∀ i = {1,2, ..., n}.

Define a distance matrix D representing the distance between each location l_k & l_m for k,m ∈ {1,2,...,n}:

$$D = \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,n} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n,1} & d_{n,2} & \cdots & d_{n,n} \end{pmatrix}$$

Where $d_{k,k} = 0$ for ∀ k = {1,2, ..., n}.

The cost function C can then be defined as:

$$C = \min_{\phi \in \xi_n} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{i,j} \cdot \phi(w_{i,j})$$

Where $w_{i,j}$ is the flow between facility f_i&f_j and $\phi(w_{i,j})$ is the distance mapping between f_i and f_j; ξ_n is the set of all possible combinations between facilities.

The travelling salesman problem is a specific case of the QAP where $w_{i,j}$ = c for some

constant c. Therefore, in this simplification, the cost function is minimised according to the sum of the distances between facilities.

From the viewpoint of commercial application, the QAP can be described as locating a number of factories (facilities) with certain manufacturing and logistic capacity (flow) such that the total distance that 'goods' produced by factories have to travel is minimised.

The QAP is an NP hard problem. Therefore, there does not exist an algorithm that can solve the problem in polynomial time. For a QAP with n facilities/locations, there exists n! possible solutions.

3 Parallel computing

A type of computation when many calculations or execution of processes are conducted simultaneously is called parallel computing (Culler, Singh, and Gupta, 1999). In parallel computing, a computational task is typically broken down into several similar sub-tasks that can be processed independently and whose results are combined afterwards, upon completion. Parallel computing is closely linked to concurrent computing, which are frequently used together, and often overlapped, though these two methods are distinct. Concurrent programming refers to environments in which the tasks we define can occur in any order. One task can occur before or after another, and all tasks can be performed simultaneously; parallel computing specifically refers to the simultaneous execution of concurrent tasks on different processors. Thus, all parallel programming is concurrent, but not all concurrent programming is parallel (BUTTLAR et al., 1996).

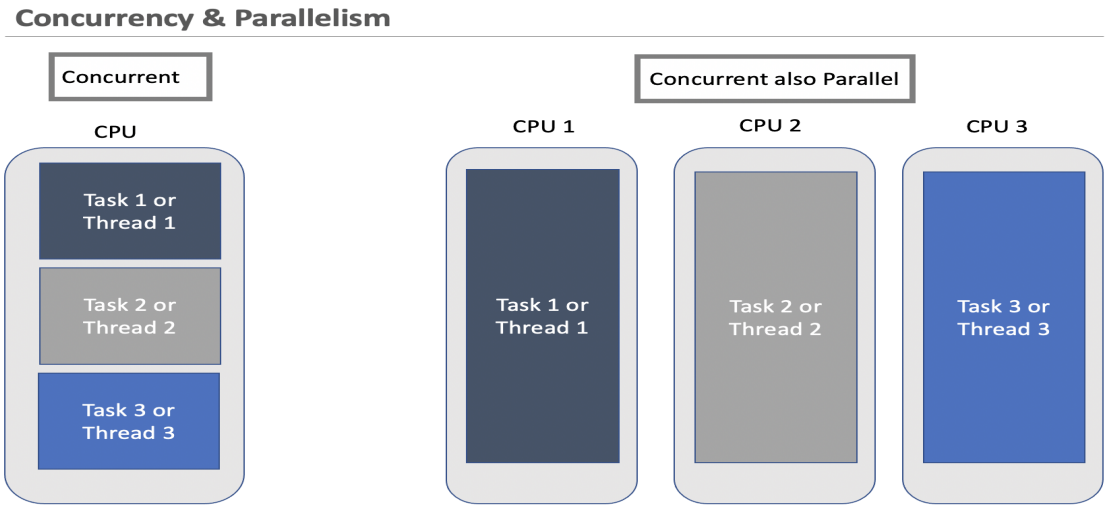


Figure 1: Concurrency & Parallelism

4 Tabu search

4.1 How it works

Tabu search is divided into four phases: initialisation phase, preliminary search phase, exploitation phase and exploration phase. These phases use various parameter settings which govern tabu list management, transition between phases and long and intermediate term memories.

The initialisation phase consists of generating a random solution $\vec{x_0}$ of size n (i.e., number of facilities). Furthermore, a customised distance matrix D and a randomised flow matrix W (with the flow between each facility ranging from 1 to 10) are generated. Lastly, the length of tabu list L is pre-defined.

In the preliminary search phase, all possible surrounding neighbourhood solutions of $\vec{x_0}$ are generated and the value of their cost functions are evaluated. Specifically in regard to our implementation of the QAP, a single neighbourhood solution is generated by swapping two facilities in the initial solution $\vec{x_0}$. Therefore, the set of neighbourhood solutions is the total combinations of any two facilities swapping in the initial solution $\vec{x_0}$ (i.e., if $\vec{x_0}$ has 8 facilities, the neighbourhood solution set would contain 28 neighbour solutions). Neighbourhood solutions are then sorted from the lowest to highest cost (in our specific implementation). The best neighbourhood solution $\vec{x_j}$ is the solution with the lowest cost and which is non-tabu (is not already listed in the tabu list). The best solution is then appended to the tabu list and is added to the full solution list. The dynamic tabu list stores the best solutions from previous iterations, which reflects the short-term memory of TS. The tabu list is the most essential part of TS because it helps to avoid cycling and premature convergence to a local optima. If the length of tabu list is greater than the pre-defined length of tabu list, the oldest solution would be removed at each iteration.

Furthermore, each iteration of the preliminary search phase evaluates the best overall solution. This 'best' solution is the best solution from the current neighbourhood solution set or the best tabu solution (aspiration citation is satisfied). If the overall best solution is from the tabu list/the aspiration citation is satisfied for a consecutive number of iterations equal to the length of the tabu list, the exploitation phase is started.

Inspired by Chakrapani and Skorin-Kapov (1993), the exploitation stage begins from the best solution found in the tabu list. The algorithm proceeds as usual in the preliminary search phase. If no better solution is found after a number of iterations equal to the pre-defined length of the tabu list L, the exploitation phase is then restarted. The exploitation phase provides a simple way to focus the search around the current best solution. This phase

reflects the intermediate-term memory of TS.

To avoid cycling and premature convergence to a local optima, we introduce the exploration phase. It diversifies our search space by exploring different areas, unrelated to the current search area. An exploration phase occurs every 25 iterations. The exploration phase starts by randomly generating three numbers in sequence with an equal gap (i.e., if $\vec{x_0}$ had a length of 8, the three numbers generated would be 2, 4, 6; length of 14 would result in 4, 8, 12). A new $\vec{x_0}$ is generated by swapping the location of three facilities in turn according to these three numbers. This phase reflects the long-term memory of TS. Following the exploration phase, the solutions in the tabu list are rearranged from best to worst and the pre-defined length of the tabu list is changed randomly to a length between 5 - 20.

Finally, we obtain the best solution with the lowest cost function value for all iterations from the full solution list. The total number of iterations must be pre-specified (the total number of iterations in our algorithm is 500).

4.2 Parallel tabu search

We implement parallel computing in TS at three different stages. At the first stage, parallel computing is implemented in the preliminary search phase when compiling the neighbourhood solution sets of our start solution $\vec{x_0}$. Similarly, during the second stage, we employ parallel computing when computing the cost of all neighbourhood solutions. Finally, different to the first and second stage where parallel computing is used to solve the same task with different input values, parallel computing could also be applied when solving two different tasks (also called concurrent computing). Therefore, we implement parallel computing when exploring different regions and modifying the tabu list (reordering tabu list and changing the length of tabu list) simultaneously. The pros and cons of parallel tabu search would be presented in section 7 supported by simulation results.

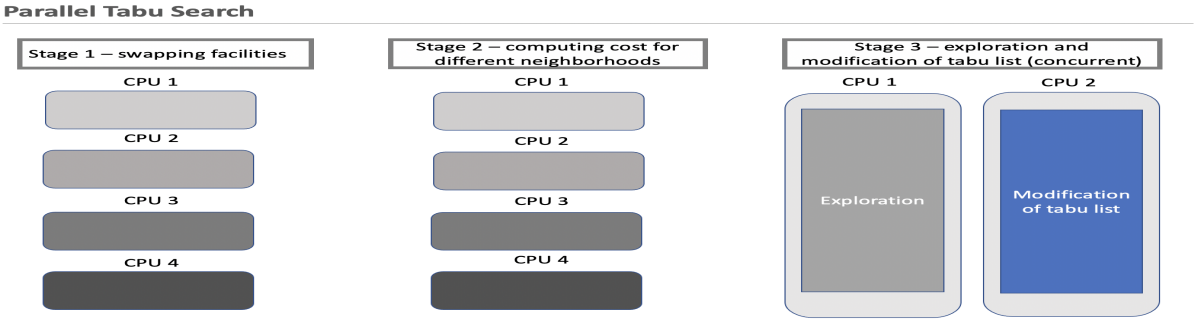


Figure 2: Parallel tabu search

5 Flow Chart

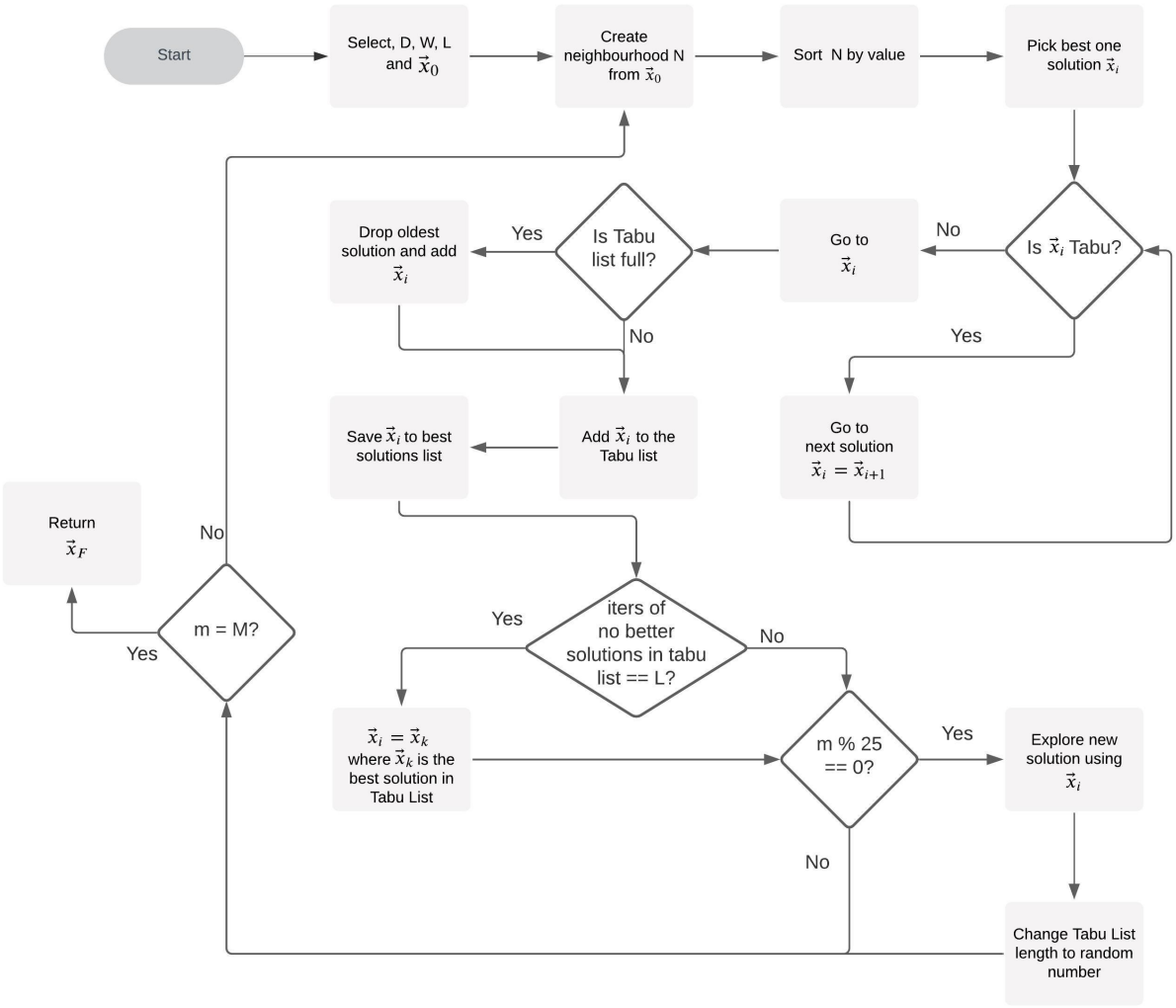


Figure 3: Tabu Search Flow Chart

6 Pseudo Code

Algorithm 1 Tabu search algorithm for solving the quadratic assignment problems

Initialisation phase

Initialise M (number of iterations), D (distance matrix), W (flow matrix), L (pre-defined length of tabu list), $\vec{x}_0$ (randomised initial solution).

For i in 1,...,M:

Preliminary search phase

1. **Generate** all possible neighborhood solutions $\vec{x}_j$ of $\vec{x}_0$ by swapping any two of facilities.

2. **Evaluate** all neighborhood solutions $\vec{x}_j$ by computing $C_j = \min_{\phi \in \xi_n} \sum_{i=1}^{n} \sum_{k=1}^{n} w_{i,k} \cdot \phi(w_{i,k})$,

where $w_{i,k}$ is the flow between f_i and f_k; $\phi(w_{i,k})$ is the distance mapping between f_i and f_k; ξ_n is the set of all possible combinations between any two facilities.

3. **Sort** C_j from best to worst.

4. **Append** the best non-tabu solution $C_j \& \vec{x}_j$ to tabu list.

5. **Remove** oldest solutions when the length of tabu list > L.

6. **Save** $C_j \& \vec{x}_j$ to full solution list for each iteration.

Exploitation phase (triggers when the iterations of no better solution found in tabu list equal to the length of the pre-defined tabu list)

1. **Assign** the best solution $\vec{x}_k$ found in tabu list as a new $\vec{x}_0$.

Exploration phase (triggers for every 25 iterations)

1. **Generate** three numbers with equal gap randomly.

2. **Create** a new $\vec{x}_0$ by swapping location of three facilities in sequence according to these three numbers.

3. **Reorder** the solutions in tabu list from best to worst.

4. **Change** L randomly, ranging from 5 to 20.

Obtain the best solution $(C_k \& \vec{x}_k)$ for all iterations from the full solution list.

7 Simulation Results

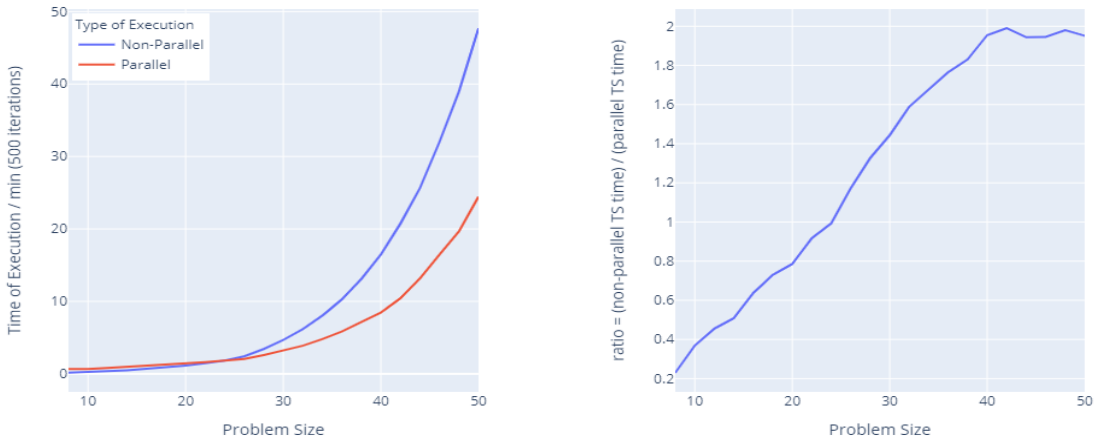
In this section, we will initialise D (customised distance matrix), F (randomised flow matrix), $\vec{x_0}$ (randomised initial solution) for varying size of n (number of facilities to be allocated), where n $\in$ (8, 10, ..., 48, 50) - in order to compare the time taken for parallel TS and non-parallel TS to solve quadratic assignment problems. The pre-defined length of tabu list L is set at 10. For both non-parallel TS and parallel TS, we run 500 iterations for various size of n. Simulation results are obtained in the table below.

Table 1: Simulation Results

500 iterations	non-parallel TS		parallel TS	
n	time(minute)	lowest cost	time(minute)	lowest cost
8	0.152	549	0.661	549
10	0.247	1000	0.668	1000
12	0.377	1368	0.828	1368
14	0.5	2370	0.983	2375
16	0.698	3262	1.095	3281
18	0.89	4817	1.22	4821
20	1.148	6574	1.462	6574
22	1.474	10000	1.607	10000
24	1.824	10738	1.838	10738
26	2.407	14610	2.054	14610
28	3.424	18340	2.58	18299
30	4.66	22074	3.229	22209
32	6.156	26579	3.875	26728
34	8.043	32484	4.794	32639
36	10.279	38357	5.825	38554
38	13.082	43933	7.147	43933
40	16.494	51947	8.439	52121
42	20.726	58242	10.416	58242
44	25.621	71058	13.183	71085
46	32.003	77236	16.452	77205
48	38.954	89947	19.681	90188
50	47.702	101582	24.447	101604

From the table above, we can find that the optimal time complexity of non-parallel TS starts to overtake parallel TS when n >= 26. This phenomenon could be explained by the fact that when the scale of QAP problems are small, the computation time itself is almost negligible when compared with allocating tasks to different CPUs to conduct parallel computing. As the figure 4(a) below shown, when the size of n gets larger than 26, the benefit of parallel computing starts to become more and more significant. Therefore, parallel TS shows superiority, in time complexity, over non-parallel TS when dealing with large-scale combinational problems.

When n <= 26, we can find that the optimal costs obtained by the two methods are almost always the same. When n > 26, we find that parallel TS and non-parallel TS obtain different optimal solutions except when n is equal to 38 or 42. These differences in optimal cost might be due to the randomness involved at the exploration phase.



(a) Time against Problem Size **(b)** Ratio of Execution Time against Size

Figure 4: Plots for Simulation Results

Figure 4(b) presents the ratio between the time complexity of non-parallel TS and parallel TS. When 8 <= n <= 40, we can observe that the efficiency of parallel TS enhances almost linearly; when n is greater than 40, the efficiency of parallel TS reaches a plateau - twice as good as non-parallel TS in terms of time complexity. As we haven't conducted simulations for n > 50 due to the limitation of our computing power, along with the fact that we initialise our simulation settings randomly, we can not conclude that parallel TS reaches its optimality when n is from 40 to 50. In an ideal scenario (i.e, unlimited computing power), we anticipate parallel computing provides exponentially increasing time benefits when n gets larger.

8 Future Improvement

In our experiment, we implemented parallel computing at three different stages, but we don't assess parallelisation is indeed effective at each stage. To obtain the answer we could execute parallel computing one stage at a time and compare the efficiency of for all stages.

As mentioned above, when n is greater than 40, the efficiency of parallel TS reaches a plateau. Further study could be done to investigate whether this trend continues for n>50. This could be performed on machines with greater computing power than was available to us.

The lowest cost for parallel and non-parallel simulations differ in the majority of simulations when n>26. Moreover, the majority of the better solutions for each n appear to belong to the non-parallel TS. Firstly, we hypothesise that optimal results for parallel and non-parallel TS would be more likely to be the same for a given n if we increased the number of total iterations in our algorithm from 500 to a large number. However, we are unsure whether the observation suggesting that non-parallel TS produces lower cost solutions is statistically significant as we only run one simulation for each n. This could be investigated further by running multiple simulations at various levels of n.

9 Conclusion

In the introduction, we hypothesised that a parallel TS algorithm would result in exponentially increasing time benefits versus a non-parallised TS as the complexity of the QAP increased.

Our results confirmed that for QAP with 8 <= n <= 40 the parallel TS does provide increasing time benefits over non-parallel TS. However, the time benefits were linearly increasing (not exponentially) and seemed to stop offering benefits when n>40. Furthermore, we found that the non-parallel TS has greater time efficiency for n<26. We believe that this is a result of the added time needed to allocate tasks to different CPU's in the case of parallel TS.

The QAP is NP hard. Therefore, it is not guaranteed that the optimal solutions generated by either the parallel or non-parallel TS will result in the global optimal solution for a finite number of total iterations (unless the total number of iterations > n!). However, although the non-parallel TS took longer than the parallel TS when the n>=26, it seemed to provide better solutions. This must be investigated further.

Consequently, the parallel TS does run faster than the non-parallel TS for QAP problems of increasing complexity. However, these time benefits may be eclipsed by the fact that the non-parallel TS may provide better solutions (although this must be investigated further).

Therefore, depending on the value of time versus accuracy in the context of the QAP problems being run, either the parallel or non-parallel TS may be the preferred algorithm.

References

[1] Rainer E Burkard et al. “The quadratic assignment problem”. In: *Handbook of combinatorial optimization*. Springer, 1998, pp. 1713–1809.

[2] DICK AUTOR BUTTLAR et al. *Pthreads programming: A POSIX standard for better multiprocessing*. " O’Reilly Media, Inc.", 1996.

[3] Jaishankar Chakrapani and Jadranka Skorin-Kapov. “Massively parallel tabu search for the quadratic assignment problem”. In: *Annals of Operations Research* 41.4 (1993), pp. 327–341.

[4] David Culler, Jaswinder Pal Singh, and Anoop Gupta. *Parallel computer architecture: a hardware/software approach*. Gulf Professional Publishing, 1999.

[5] C-N Fiechter. “A parallel tabu search algorithm for large traveling salesman problems”. In: *Discrete Applied Mathematics* 51.3 (1994), pp. 243–267.

10 Appendix

10.1 Python history

_____

_____

Size of n: 8

_____

_____

Non-parellel Tabu Search
Min in all Iterations: ['549' 'A2' 'A0' 'A6' 'A7' 'A5' 'A1' 'A4' 'A3']
The Lowest Cost is: 549
Total time it takes: 0.152 minutes.
Parellel Tabu Search
Min in all Iterations: ['549' 'A2' 'A0' 'A6' 'A7' 'A5' 'A1' 'A4' 'A3']
The Lowest Cost is: 549
Total time it takes: 0.661 minutes.

_____

_____

Size of n: 10

_____

_____

Non-parellel Tabu Search
Min in all Iterations: ['1000' 'A6' 'A2' 'A5' 'A3' 'A9' 'A1' 'A0' 'A7' 'A4' 'A8']
The Lowest Cost is: 1000
Total time it takes: 0.247 minutes.
Parellel Tabu Search
Min in all Iterations: ['1000' 'A6' 'A2' 'A5' 'A3' 'A9' 'A1' 'A0' 'A7' 'A4' 'A8']
The Lowest Cost is: 1000
Total time it takes: 0.668 minutes.

_____

_____

Size of n: 12

_____

_____

Non-parellel Tabu Search

Min in all Iterations: ['1368' 'A9' 'A4' 'A7' 'A5' 'A6' 'A10' 'A3' 'A11' 'A1' 'A2' 'A0' 'A8']
The Lowest Cost is: 1368
Total time it takes: 0.377 minutes.
Parellel Tabu Search
Min in all Iterations: ['1368' 'A9' 'A4' 'A7' 'A5' 'A6' 'A10' 'A3' 'A11' 'A1' 'A2' 'A0' 'A8']
The Lowest Cost is: 1368
Total time it takes: 0.828 minutes.
_____
_____
Size of n: 14
_____
_____
Non-parellel Tabu Search
Min in all Iterations: ['2370' 'A2' 'A13' 'A6' 'A1' 'A11' 'A7' 'A0' 'A12' 'A3' 'A9' 'A5' 'A4' 'A8' 'A10']
The Lowest Cost is: 2370
Total time it takes: 0.5 minutes.
Parellel Tabu Search
Min in all Iterations: ['2375' 'A12' 'A13' 'A6' 'A5' 'A11' 'A7' 'A0' 'A2' 'A3' 'A9' 'A1' 'A4' 'A8' 'A10']
The Lowest Cost is: 2375
Total time it takes: 0.983 minutes.
_____
_____
Size of n: 16
_____
_____
Non-parellel Tabu Search
Min in all Iterations: ['3262' 'A8' 'A2' 'A14' 'A12' 'A5' 'A3' 'A4' 'A7' 'A13' 'A9' 'A1' 'A10' 'A15' 'A6' 'A11' 'A0']
The Lowest Cost is: 3262
Total time it takes: 0.698 minutes.
Parellel Tabu Search
Min in all Iterations: ['3281' 'A9' 'A1' 'A14' 'A2' 'A5' 'A3' 'A15' 'A8' 'A13' 'A12' 'A10' 'A11' 'A4' 'A6' 'A7' 'A0']

The Lowest Cost is: 3281
Total time it takes: 1.095 minutes.
_____
_____
Size of n: 18
_____
_____
Non-parellel Tabu Search
Min in all Iterations: ['4817' 'A2' 'A4' 'A1' 'A16' 'A15' 'A6' 'A9' 'A0' 'A11' 'A14' 'A8' 'A3' 'A12' 'A13' 'A17' 'A10' 'A5' 'A7']
The Lowest Cost is: 4817
Total time it takes: 0.89 minutes.
Parellel Tabu Search
Min in all Iterations: ['4821' 'A4' 'A3' 'A1' 'A16' 'A15' 'A6' 'A9' 'A0' 'A11' 'A2' 'A14' 'A8' 'A12' 'A13' 'A17' 'A10' 'A5' 'A7']
The Lowest Cost is: 4821
Total time it takes: 1.22 minutes.
_____
_____
Size of n: 20
_____
_____
Non-parellel Tabu Search
Min in all Iterations: ['6574' 'A5' 'A4' 'A2' 'A17' 'A13' 'A10' 'A19' 'A14' 'A18' 'A15' 'A11' 'A12' 'A0' 'A3' 'A16' 'A1' 'A7' 'A8' 'A9' 'A6']
The Lowest Cost is: 6574
Total time it takes: 1.148 minutes.
Parellel Tabu Search
Min in all Iterations: ['6574' 'A5' 'A4' 'A2' 'A17' 'A13' 'A10' 'A19' 'A14' 'A18' 'A15' 'A11' 'A12' 'A0' 'A3' 'A16' 'A1' 'A7' 'A8' 'A9' 'A6']
The Lowest Cost is: 6574
Total time it takes: 1.462 minutes.
_____
_____
Size of n: 22

Non-parellel Tabu Search
Min in all Iterations: ['10000' 'A3' 'A9' 'A14' 'A5' 'A15' 'A18' 'A1' 'A10' 'A7' 'A19' 'A2' 'A13' 'A8' 'A21' 'A11' 'A16' 'A6' 'A20' 'A0' 'A12' 'A4' 'A17']
The Lowest Cost is: 10000
Total time it takes: 1.474 minutes.
Parellel Tabu Search
Min in all Iterations: ['10000' 'A3' 'A9' 'A14' 'A5' 'A15' 'A18' 'A1' 'A10' 'A7' 'A19' 'A2' 'A13' 'A8' 'A21' 'A11' 'A16' 'A6' 'A20' 'A0' 'A12' 'A4' 'A17']
The Lowest Cost is: 10000
Total time it takes: 1.607 minutes.

Size of n: 24

Non-parellel Tabu Search
Min in all Iterations: ['10738' 'A9' 'A18' 'A8' 'A4' 'A15' 'A21' 'A7' 'A2' 'A11' 'A12' 'A0' 'A10' 'A17' 'A13' 'A22' 'A23' 'A3' 'A16' 'A1' 'A20' 'A6' 'A14' 'A19' 'A5']
The Lowest Cost is: 10738
Total time it takes: 1.824 minutes.
Parellel Tabu Search
Min in all Iterations: ['10738' 'A9' 'A18' 'A8' 'A4' 'A15' 'A21' 'A7' 'A2' 'A11' 'A12' 'A0' 'A10' 'A17' 'A13' 'A22' 'A23' 'A3' 'A16' 'A1' 'A20' 'A6' 'A14' 'A19' 'A5']
The Lowest Cost is: 10738
Total time it takes: 1.838 minutes.

Size of n: 26

Non-parellel Tabu Search
Min in all Iterations: ['14610' 'A10' 'A24' 'A22' 'A21' 'A4' 'A17' 'A12' 'A18' 'A16' 'A25' 'A3' 'A9' 'A1' 'A14' 'A13' 'A5' 'A7' 'A0' 'A15' 'A23' 'A19' 'A20' 'A11' 'A6'

'A8' 'A2']
The Lowest Cost is: 14610
Total time it takes: 2.407 minutes.
Parellel Tabu Search
Min in all Iterations: ['14610' 'A10' 'A24' 'A22' 'A21' 'A4' 'A17' 'A12' 'A18' 'A16' 'A25' 'A3'
'A9' 'A1' 'A14' 'A13' 'A5' 'A7' 'A0' 'A15' 'A23' 'A19' 'A20' 'A11' 'A6'
'A8' 'A2']
The Lowest Cost is: 14610
Total time it takes: 2.054 minutes.
_____
_____
Size of n: 28
_____
_____
Non-parellel Tabu Search
Min in all Iterations: ['18340' 'A16' 'A0' 'A27' 'A23' 'A7' 'A21' 'A19' 'A2' 'A20' 'A1' 'A8' 'A9'
'A25' 'A5' 'A13' 'A15' 'A18' 'A10' 'A14' 'A24' 'A12' 'A3' 'A4' 'A17'
'A11' 'A22' 'A6' 'A26']
The Lowest Cost is: 18340
Total time it takes: 3.424 minutes.
Parellel Tabu Search
Min in all Iterations: ['18299' 'A16' 'A0' 'A27' 'A23' 'A7' 'A24' 'A21' 'A2' 'A20' 'A1' 'A8' 'A9'
'A25' 'A22' 'A13' 'A15' 'A18' 'A10' 'A14' 'A12' 'A3' 'A19' 'A4' 'A17'
'A5' 'A11' 'A26' 'A6']
The Lowest Cost is: 18299
Total time it takes: 2.58 minutes.
_____
_____
Size of n: 30
_____
_____
Non-parellel Tabu Search
Min in all Iterations: ['22074' 'A4' 'A3' 'A2' 'A15' 'A5' 'A27' 'A10' 'A25' 'A17' 'A29' 'A11'
'A19' 'A18' 'A20' 'A21' 'A9' 'A7' 'A26' 'A0' 'A8' 'A23' 'A12' 'A24' 'A6'
'A28' 'A22' 'A13' 'A1' 'A16' 'A14']

The Lowest Cost is: 22074
Total time it takes: 4.66 minutes.
Parellel Tabu Search
Min in all Iterations: ['22209' 'A4' 'A3' 'A15' 'A10' 'A24' 'A27' 'A5' 'A25' 'A17' 'A29' 'A11'
'A20' 'A18' 'A19' 'A1' 'A9' 'A7' 'A26' 'A28' 'A12' 'A8' 'A23' 'A2' 'A6'
'A0' 'A22' 'A13' 'A21' 'A14' 'A16']
The Lowest Cost is: 22209
Total time it takes: 3.229 minutes.

Size of n: 32

Non-parellel Tabu Search
Min in all Iterations: ['26579' 'A10' 'A14' 'A11' 'A1' 'A26' 'A8' 'A22' 'A24' 'A27' 'A4' 'A0'
'A6' 'A23' 'A25' 'A5' 'A13' 'A17' 'A31' 'A29' 'A30' 'A18' 'A12' 'A3' 'A9'
'A7' 'A16' 'A20' 'A15' 'A21' 'A19' 'A28' 'A2']
The Lowest Cost is: 26579
Total time it takes: 6.156 minutes.
Parellel Tabu Search
Min in all Iterations: ['26728' 'A14' 'A11' 'A29' 'A18' 'A16' 'A27' 'A8' 'A6' 'A4' 'A20' 'A0'
'A15' 'A23' 'A25' 'A28' 'A2' 'A17' 'A31' 'A30' 'A1' 'A26' 'A12' 'A10'
'A9' 'A3' 'A24' 'A22' 'A7' 'A21' 'A19' 'A5' 'A13']
The Lowest Cost is: 26728
Total time it takes: 3.875 minutes.

Size of n: 34

Non-parellel Tabu Search
Min in all Iterations: ['32484' 'A21' 'A9' 'A0' 'A4' 'A7' 'A19' 'A31' 'A16' 'A33' 'A32' 'A23'
'A10' 'A8' 'A11' 'A18' 'A30' 'A5' 'A28' 'A2' 'A24' 'A29' 'A3' 'A15' 'A13'
'A14' 'A6' 'A12' 'A1' 'A20' 'A17' 'A25' 'A26' 'A27' 'A22']
The Lowest Cost is: 32484

Total time it takes: 8.043 minutes.
Parellel Tabu Search
Min in all Iterations: ['32639' 'A21' 'A29' 'A9' 'A25' 'A7' 'A33' 'A20' 'A23' 'A12' 'A1' 'A19'
'A10' 'A8' 'A11' 'A30' 'A18' 'A5' 'A2' 'A24' 'A0' 'A3' 'A13' 'A31' 'A32'
'A16' 'A6' 'A14' 'A4' 'A15' 'A17' 'A26' 'A27' 'A28' 'A22']
The Lowest Cost is: 32639
Total time it takes: 4.794 minutes.

Size of n: 36

Non-parellel Tabu Search
Min in all Iterations: ['38357' 'A6' 'A30' 'A16' 'A24' 'A14' 'A22' 'A8' 'A0' 'A1' 'A11' 'A13'
'A23' 'A4' 'A29' 'A26' 'A28' 'A35' 'A32' 'A18' 'A15' 'A31' 'A27' 'A7'
'A9' 'A33' 'A25' 'A17' 'A21' 'A3' 'A19' 'A12' 'A5' 'A2' 'A10' 'A20' 'A34']
The Lowest Cost is: 38357
Total time it takes: 10.279 minutes.
Parellel Tabu Search
Min in all Iterations: ['38554' 'A18' 'A30' 'A16' 'A24' 'A9' 'A33' 'A22' 'A25' 'A8' 'A32' 'A3'
'A28' 'A4' 'A34' 'A15' 'A2' 'A31' 'A20' 'A6' 'A27' 'A7' 'A11' 'A14' 'A1'
'A0' 'A17' 'A23' 'A21' 'A13' 'A29' 'A19' 'A10' 'A12' 'A26' 'A5' 'A35']
The Lowest Cost is: 38554
Total time it takes: 5.825 minutes.

Size of n: 38

Non-parellel Tabu Search
Min in all Iterations: ['43933' 'A21' 'A26' 'A15' 'A22' 'A20' 'A5' 'A18' 'A4' 'A35' 'A12' 'A37'
'A14' 'A36' 'A19' 'A24' 'A16' 'A27' 'A28' 'A2' 'A8' 'A9' 'A1' 'A0' 'A29'
'A13' 'A34' 'A3' 'A17' 'A10' 'A30' 'A32' 'A25' 'A23' 'A7' 'A31' 'A6'
'A33' 'A11']
The Lowest Cost is: 43933

Total time it takes: 13.082 minutes.
Parellel Tabu Search
Min in all Iterations: ['43933' 'A21' 'A26' 'A15' 'A22' 'A20' 'A5' 'A18' 'A4' 'A35' 'A12' 'A37'
'A14' 'A36' 'A19' 'A24' 'A16' 'A27' 'A28' 'A2' 'A8' 'A9' 'A1' 'A0' 'A29'
'A13' 'A34' 'A3' 'A17' 'A10' 'A30' 'A32' 'A25' 'A23' 'A7' 'A31' 'A6'
'A33' 'A11']
The Lowest Cost is: 43933
Total time it takes: 7.147 minutes.

Size of n: 40

Non-parellel Tabu Search
Min in all Iterations: ['51947' 'A37' 'A6' 'A25' 'A0' 'A9' 'A35' 'A26' 'A34' 'A14' 'A39' 'A5'
'A11' 'A7' 'A12' 'A29' 'A17' 'A27' 'A32' 'A4' 'A15' 'A20' 'A8' 'A21'
'A24' 'A30' 'A18' 'A38' 'A2' 'A22' 'A33' 'A13' 'A36' 'A31' 'A19' 'A3'
'A1' 'A28' 'A10' 'A16' 'A23']
The Lowest Cost is: 51947
Total time it takes: 16.494 minutes.
Parellel Tabu Search
Min in all Iterations: ['52121' 'A6' 'A25' 'A30' 'A38' 'A35' 'A2' 'A26' 'A34' 'A14' 'A5' 'A13'
'A11' 'A7' 'A12' 'A3' 'A17' 'A29' 'A27' 'A10' 'A15' 'A20' 'A21' 'A24'
'A8' 'A18' 'A0' 'A9' 'A37' 'A22' 'A33' 'A39' 'A36' 'A31' 'A32' 'A28'
'A19' 'A1' 'A16' 'A4' 'A23']
The Lowest Cost is: 52121
Total time it takes: 8.439 minutes.

Size of n: 42

Non-parellel Tabu Search
Min in all Iterations: ['58242' 'A1' 'A32' 'A33' 'A2' 'A0' 'A3' 'A5' 'A38' 'A21' 'A34' 'A39' 'A6'
'A17' 'A27' 'A22' 'A9' 'A25' 'A18' 'A4' 'A41' 'A12' 'A19' 'A24' 'A36'

'A16' 'A11' 'A8' 'A30' 'A26' 'A40' 'A15' 'A37' 'A31' 'A13' 'A23' 'A35'
'A10' 'A14' 'A28' 'A20' 'A7' 'A29']
The Lowest Cost is: 58242
Total time it takes: 20.726 minutes.
Parellel Tabu Search
Min in all Iterations: ['58242' 'A1' 'A32' 'A33' 'A2' 'A0' 'A3' 'A5' 'A38' 'A21' 'A34' 'A39' 'A6'
'A17' 'A27' 'A22' 'A9' 'A25' 'A18' 'A4' 'A41' 'A12' 'A19' 'A24' 'A36'
'A16' 'A11' 'A8' 'A30' 'A26' 'A40' 'A15' 'A37' 'A31' 'A13' 'A23' 'A35'
'A10' 'A14' 'A28' 'A20' 'A7' 'A29']
The Lowest Cost is: 58242
Total time it takes: 10.416 minutes.

Size of n: 44

Non-parellel Tabu Search
Min in all Iterations: ['71058' 'A34' 'A3' 'A1' 'A37' 'A36' 'A13' 'A14' 'A2' 'A27' 'A42' 'A26'
'A8' 'A7' 'A31' 'A15' 'A33' 'A11' 'A0' 'A30' 'A23' 'A9' 'A6' 'A29' 'A28'
'A39' 'A5' 'A40' 'A17' 'A43' 'A24' 'A4' 'A16' 'A25' 'A10' 'A18' 'A22'
'A41' 'A21' 'A12' 'A32' 'A35' 'A38' 'A20' 'A19']
The Lowest Cost is: 71058
Total time it takes: 25.621 minutes.
Parellel Tabu Search
Min in all Iterations: ['71085' 'A34' 'A28' 'A37' 'A1' 'A36' 'A13' 'A14' 'A24' 'A2' 'A42' 'A26'
'A8' 'A7' 'A31' 'A15' 'A33' 'A11' 'A0' 'A30' 'A23' 'A6' 'A19' 'A29' 'A3'
'A5' 'A39' 'A40' 'A4' 'A43' 'A17' 'A27' 'A16' 'A25' 'A10' 'A18' 'A22'
'A41' 'A21' 'A12' 'A32' 'A35' 'A38' 'A20' 'A9']
The Lowest Cost is: 71085
Total time it takes: 13.183 minutes.

Size of n: 46

Non-parellel Tabu Search
Min in all Iterations: ['77236' 'A21' 'A4' 'A7' 'A10' 'A25' 'A39' 'A14' 'A30' 'A43' 'A27' 'A8'
'A32' 'A1' 'A35' 'A26' 'A5' 'A38' 'A28' 'A15' 'A40' 'A9' 'A34' 'A45' 'A6'
'A17' 'A3' 'A36' 'A20' 'A0' 'A37' 'A24' 'A33' 'A42' 'A23' 'A44' 'A31'
'A22' 'A19' 'A41' 'A12' 'A18' 'A2' 'A16' 'A11' 'A29' 'A13']
The Lowest Cost is: 77236
Total time it takes: 32.003 minutes.
Parellel Tabu Search
Min in all Iterations: ['77205' 'A21' 'A10' 'A7' 'A4' 'A25' 'A39' 'A14' 'A30' 'A43' 'A27' 'A8'
'A32' 'A1' 'A35' 'A26' 'A5' 'A38' 'A28' 'A9' 'A40' 'A34' 'A29' 'A45' 'A6'
'A17' 'A36' 'A3' 'A20' 'A0' 'A37' 'A24' 'A33' 'A42' 'A23' 'A44' 'A31'
'A22' 'A19' 'A41' 'A12' 'A18' 'A2' 'A15' 'A16' 'A11' 'A13']
The Lowest Cost is: 77205
Total time it takes: 16.452 minutes.

Size of n: 48

Non-parellel Tabu Search
Min in all Iterations: ['89947' 'A22' 'A1' 'A45' 'A14' 'A32' 'A40' 'A39' 'A11' 'A19' 'A30' 'A36'
'A31' 'A6' 'A37' 'A12' 'A17' 'A20' 'A21' 'A28' 'A13' 'A26' 'A25' 'A0'
'A18' 'A41' 'A38' 'A8' 'A35' 'A10' 'A2' 'A46' 'A9' 'A16' 'A5' 'A29' 'A7'
'A3' 'A42' 'A27' 'A4' 'A23' 'A24' 'A44' 'A43' 'A47' 'A34' 'A15' 'A33']
The Lowest Cost is: 89947
Total time it takes: 38.954 minutes.
Parellel Tabu Search
Min in all Iterations: ['90188' 'A22' 'A1' 'A45' 'A14' 'A32' 'A40' 'A39' 'A11' 'A19' 'A44' 'A36'
'A31' 'A7' 'A29' 'A27' 'A12' 'A17' 'A21' 'A24' 'A0' 'A26' 'A25' 'A34'
'A18' 'A41' 'A38' 'A8' 'A35' 'A10' 'A2' 'A30' 'A15' 'A5' 'A9' 'A16' 'A6'
'A3' 'A42' 'A46' 'A37' 'A28' 'A23' 'A20' 'A4' 'A13' 'A43' 'A47' 'A33']
The Lowest Cost is: 90188
Total time it takes: 19.681 minutes.

Size of n: 50

Non-parellel Tabu Search
Min in all Iterations: ['101582' 'A5' 'A42' 'A8' 'A2' 'A9' 'A18' 'A1' 'A13' 'A23' 'A27' 'A17'
'A46' 'A10' 'A19' 'A0' 'A14' 'A24' 'A48' 'A26' 'A35' 'A4' 'A31' 'A49'
'A43' 'A45' 'A39' 'A41' 'A33' 'A37' 'A16' 'A44' 'A38' 'A34' 'A12' 'A7'
'A20' 'A29' 'A25' 'A6' 'A40' 'A47' 'A22' 'A36' 'A21' 'A3' 'A32' 'A11'
'A30' 'A28' 'A15']
The Lowest Cost is: 101582
Total time it takes: 47.702 minutes.
Parellel Tabu Search
Min in all Iterations: ['101604' 'A5' 'A39' 'A8' 'A16' 'A2' 'A18' 'A1' 'A13' 'A23' 'A27' 'A17'
'A10' 'A46' 'A40' 'A24' 'A14' 'A48' 'A26' 'A35' 'A32' 'A4' 'A11' 'A31'
'A43' 'A45' 'A42' 'A41' 'A44' 'A33' 'A37' 'A9' 'A38' 'A34' 'A12' 'A20'
'A7' 'A19' 'A25' 'A29' 'A0' 'A6' 'A47' 'A36' 'A22' 'A3' 'A21' 'A30' 'A49'
'A28' 'A15']
The Lowest Cost is: 101604
Total time it takes: 24.447 minutes.
Process finished with exit code 0