Solving the Quadratic Assignment Problem by a Hybrid Algorithm

Aldy Gunawan*, Kien Ming Ng*, Kim Leng Poh*

*School of Information Systems, Singapore Management University
80 Stamford Road, Singapore, 178902, Singapore
Tel: +65 6828 0276, E-mail: aldygunawan@smu.edu.sg

*Industrial and Systems Engineering Department, National University of Singapore
10 Kent Ridge Crescent, Singapore, 119260, Singapore
Tel: +65 6516 2193, E-mail: isenkm@nus.edu.sg and isepohkl@nus.edu.sg

Abstract—This paper presents a hybrid algorithm to solve the Quadratic Assignment Problem (QAP). The proposed algorithm involves using the Greedy Randomized Adaptive Search Procedure (GRASP) to obtain an initial solution, and then using a combined Simulated Annealing (SA) and Tabu Search (TS) algorithm to improve the solution. Experimental results indicate that the hybrid algorithm is able to obtain good quality solutions for QAPLIB test problems within reasonable computation time.

Keywords—Quadratic Assignment Problem, Hybrid Algorithm, GRASP, Simulated Annealing, Tabu Search.

I. INTRODUCTION

The Quadratic Assignment Problem (QAP) is identified as the problem of finding a minimum cost allocation of facilities into locations, with the costs being the sum of all possible distance-flow products [1]. It is a combinatorial optimization problem that is first stated by Koopmans and Beckmann [2]. This problem belongs to the class of NP-hard problems and there is no ε-approximation algorithm for the QAP unless P=NP [3].

Some of the recent surveys of the QAP in the literature were presented by Anstreicher [4], Drezner et al. [5] and Loiola et al. [1]. There are many practical problems that can be formulated as a QAP, such as problems dealing with backboard wiring [6], campus layout [7], hospital planning [8], scheduling [9] and turbine balancing [10]. The QAP can also be formulated in different ways, such as pure integer programming formulations [11,12,13], mixed integer linear programming formulations [14,15], trace formulations [16,17], graph formulations [18,19] and permutation problems [20,21].

Both exact and heuristic methods have been used to solve the QAP. Exact algorithms, which include the branch-and-bound, dynamic programming and cutting plane techniques, can only be used to solve small-size instances of the problem. Thus, many heuristics have been proposed by researchers to find optimal or near optimal solutions for the QAP. These heuristics range from simple iterative improvement procedures to metaheuristic implementations, such as Ant Colony Optimization [22,23], Genetic Algorithm [20,24,25], Tabu Search [21,26] and Simulated Annealing [13,27,28]. Loiola et al. [1] highlighted the development of hybrid algorithms for solving the QAP. These hybrid algorithms for the QAP include a combination of Tabu Search with Simulated Annealing as presented by Misevicius [29], while Youssef et al. [30] used Tabu Search, Simulated Annealing and fuzzy logic together to solve the QAP.

This paper presents a new hybrid metaheuristic for the QAP. It involves three different algorithms: GRASP (Greedy Randomized Adaptive Search Procedure), Simulated Annealing (SA) and Tabu Search (TS). An extensive computational testing of this hybrid metaheuristic has been carried out with the benchmark instances in the QAPLIB, a well-known library of QAP instance [31].

The rest of this paper is organized as follows. In Section II, we provide a description of the problems considered in this paper. In Section III, the proposed hybrid algorithm is explained in detail. The computational results of applying the hybrid algorithm are presented in Section IV, and some concluding remarks are provided in Section V.
II. PROBLEM DESCRIPTION

The QAP can be described as the assignment of n facilities to n different locations. Given two n × n matrices, \( F = [f_{ij}] \) and \( D = [d_{ij}] \), where \( f_{ij} \) is the flow between facilities \( i \) and \( j \) and \( d_{ij} \) is the distance between locations \( k \) and \( l \), the problem can be formulated as follows [1]:

\[
\text{Minimize } Z = \sum_{i,j} \sum_{k,l} f_{ij} d_{ik} x_{ij} x_{kl} \tag{1}
\]

subject to:

\[
\sum_{i,j} x_{ij} = 1 \quad 1 \leq k \leq n \tag{2}
\]

\[
\sum_{i,j} x_{ij} = 1 \quad 1 \leq i \leq n \tag{3}
\]

\[
x_{ik} \in \{0, 1\} \quad 1 \leq i, k \leq n \tag{4}
\]

The objective function represents the total cost of assignment of all facilities to all locations, which is the product of the flow between facilities \( i \) and \( j \) and the distance between locations \( k \) and \( l \). The constraints ensure that exactly one facility is assigned to each location.

The QAP can also be represented as a permutation problem. Let \( f_{ij} \) be the flow between facilities \( i \) and \( j \) and \( d_{ij}(e(j)) \) be the distance between locations \( \pi(i) \) and \( \pi(j) \).

The QAP problem then becomes:

\[
\min_{\pi \in \Pi(n)} Z(\pi) = \sum_{i,j} f_{ij} d_{ij}(\pi(i)) \tag{5}
\]

where \( \Pi(n) \) is the set of all permutations of integers \( \{1, 2, ..., n\} \).

In this paper, a solution to the QAP is represented by the vector: \( \pi = [\pi(1), \pi(2), \pi(3), ..., \pi(n)] \), where the element \( \pi(i) = k \) denotes that facility \( i \) is assigned to location \( k \).

III. THE PROPOSED ALGORITHM

The hybrid algorithm proposed in this paper comprises of two main phases: (1) construction, and (2) improvement. The GRASP algorithm is used to initialize a solution in the first phase, while a combined SA and TS (Algorithm SA-TS) is used to improve the solution in the second phase. Each phase is presented and described in detail below.

A. Construction Phase

In the construction phase, we build an initial solution by implementing part of the Greedy Randomized Adaptive Search Procedure (GRASP). The GRASP is a metaheuristic that combines constructive heuristics and local search [32]. It comprises of the two steps: solution construction and solution improvement.

In the first step, we construct an initial solution by adding one new element at a time. The process selection is initially started by building the candidate list, called restricted candidate list. An element is then picked randomly from the list. In our implementation, we only consider the first process to construct an initial solution. The construction process of GRASP is shown in Figure 1.

GRASP algorithm ()

(1) Sort the \( [n^2 - n] \) flow entries in \( F \) in increasing order and keep the largest \( \beta [n^2 - n] \) entries, such that \( f_{i,\beta} \geq f_{i,\alpha} \geq ... \geq f_{i,1} \). Let \( \beta \) be the first candidate restriction parameter (0 < \( \beta < 1 \)) and \( [x] \) be the largest integer smaller or equal to \( x \).

(2) Sort the \( [n^2 - n] \) distance entries in \( D \) in non-increasing order and keep the smallest \( \beta [n^2 - n] \) entries, such that \( d_{i,\beta} \leq d_{i,\alpha} \leq ... \leq d_{i,1} \).

(3) Calculate the cost interactions \( f_{i,\beta} d_{i,\beta} \) for \( i = 1, 2, ..., [n^2 - n] \), sort them in increasing order and keep the smallest \( [\gamma n^2 - n] \) elements as the candidate list, where \( \gamma \) is the second candidate restriction parameter (0 < \( \gamma < 1 \)).

(4) Select two elements from the candidate list randomly.

(5) Calculate \( c_{ij} \), the cost of assigning facility \( i \) to location \( k \), with respect to the already-made assignments, \( \Gamma : \gamma = \sum_{(i,j) \in \Gamma} f_{ij} d_{ij} \) where \( \Gamma = \{ (j,k), (j,k), ..., (j,k) \} \).

(6) Set \( o = n - \beta \).

(7) Determine the \( [\gamma n^2 - n] \) facility-location pairs having the smallest \( c_{ij} \) values.

(8) Select a facility-location pair \( (i,k) \) randomly from the list generated in Step 7.

(9) Update the set \( \Gamma = \Gamma \cup (i,k) \).

(10) Set \( o = o - 1 \).

(11) Repeat Steps 5 – 10 until \( o = 0 \).

Fig. 1. GRASP Algorithm

B. Improvement Phase

The initial solution generated by GRASP, initial_solution, is then improved in the improvement phase. The algorithm applied in this phase is a combined SA and TS algorithm (Algorithm SA-TS). While it is mainly based on Simulated Annealing [33], the main difference of the standard SA and the proposed SA lies in the additional elements or strategies added. Several features from Tabu Search, such as the tabu length, tabu list and the intensification strategy are incorporated in the algorithm for further improvement [34,35].

In order to improve the solution, a local search algorithm involving a partial sequential neighborhood search is also augmented. The basic idea of the search is to swap or exchange the locations of two facilities such that a better solution is derived. Assuming that \( f_{i,j} = f_{j,i} \), the objective function difference \( \Delta x(i,j) \) obtained by exchanging facilities \( x(i) \) and \( x(j) \) can be computed in \( O(n) \) operations, using the following equation [36]:

\[
\Delta x(i,j) = f_{i,j} d_{i,j}(i) - d_{i,j}(j) + f_{i,j} d_{j,i}(i) - d_{j,i}(j) + \sum_{\alpha \neq i,j} f_{\alpha,j} d_{\alpha,j}(i) - d_{\alpha,j}(j) + f_{\alpha,i} d_{\alpha,i}(j) - d_{\alpha,i}(j) \tag{6}
\]
If both matrices $F$ and $D$ are symmetric with a zero diagonal, the formula can be simplified as follows:

$$A_{F,i,j} = 2 \sum_{a=1}^{n} f_a(d_{(a,\pi(i))} - d_{(a,\pi(j))}) + f_0(d_{(0,\pi(i))} - d_{(0,\pi(j))})$$  \hspace{1cm} (7)$$

Instead of selecting two facilities randomly as was commonly done in SA, we start by selecting one facility $i$ randomly followed by examining all other potential pair-swaps sequentially in the order $[i, j; i \neq j]$. The selected move is the one with the best $A_{F,i,j}$ value. The new permutation is then evaluated by the acceptance-rejection procedure in SA.

The tabu list contains pairs $(i, j)$ that have been visited in the last $\text{length}$ iterations. For a given iteration, if a pair $(i, j)$ belongs to the tabu list, it is not allowed to accept the exchange of facilities $i$ and $j$, unless this exchange gives an objective function value strictly better than the previous one (aspiration level criteria). At any temperature $T$, the neighborhood search is repeated until a certain number of iterations, $\text{inner_loop}$, has been performed.

The details of this procedure are summarized in Figure 2.

If there is no improvement of the solution obtained within a certain number of iterations ($\text{limit}$), we apply an intensification strategy of Tabu Search. This strategy focuses the search once again starting from the best permutation obtained. Finally, the entire algorithm will be terminated if the total number of iterations of the outer loop reaches the preset maximum number of iterations, $\text{outer_loop}$.

**Algorithm SA-TS ( )**

1. Initialize the parameters
2. Set the best solution, $\text{best_sol} = \text{initial_sol}$
3. Set the current solution, $\text{current_sol} = \text{initial_sol}$
4. Set the total number of iterations, $\text{num_iter} = 0$
5. Set the total number of iterations without improvement, $\text{no_improv} = 0$
6. While the total number of iterations, $\text{num_iter}$ is less than the preset maximum number of iterations, $\text{outer_loop}$ do:
   7. Repeat $\text{inner_loop}$ times:
      8. Select a facility $i$ randomly
      9. Apply a partial sequential neighborhood search
     10. Find the best permutation $\pi'$ with the smallest value of $A_{F,i,\pi'}$
     11. Check whether the best permutation is tabu or not
      12. If $A_{F,i,\pi'} < 0$
        13. Update the current solution, $\text{current_sol}$
       14. If $\text{current_sol}$ is better than $\text{best_sol}$
        15. Update the best solution, $\text{best_sol} = \text{current_sol}$
       16. Update tabu list
      17. Else
        18. Choose a random number $\epsilon$ uniformly from $[0,1]$
        19. $\text{no_improv} = \text{no_improv} + 1$
        20. If $\epsilon < \exp(-\alpha T_{\text{current}})$ and the new solution is not tabu
        21. Accept the new solution, $\text{new_sol}$
       22. Update the current solution, $\text{current_sol}$
       23. Update tabu list
      24. Else
        25. Return to the current solution, $\text{current_sol}$
        26. Update tabu list
        27. Update temperature $T_{\text{current}} = T_{\text{current}} / 2$
        28. If ($\text{no_improv} > \text{limit}$)
        29. Apply the intensification strategy
       30. Set $\text{no_improv} = 0$
      31. $\text{num_iter} = \text{num_iter} + 1$
     32. End while
    33. Report the best solution, $\text{best_sol}$

**IV. COMPUTATIONAL RESULTS**

**A. Experimental Setup**

The values of the parameters used in the computational experiments are determined experimentally to ensure a compromise between the computation time and the solution quality. They are summarized in Table I. The algorithms were implemented using C++ and executed on a 2.67 GHz Intel Core 2 Duo CPU with 3 GB of RAM under the Microsoft Windows Vista Operating System.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of iterations, $\text{outer_loop}$</td>
<td>300n</td>
</tr>
<tr>
<td>Initial temperature, $T_1$</td>
<td>5,000</td>
</tr>
<tr>
<td>Number of neighborhood moves at each temperature $T$, $\text{inner_loop}$</td>
<td>100n</td>
</tr>
<tr>
<td>Cooling factor, $\alpha$</td>
<td>0.9</td>
</tr>
<tr>
<td>Number of non-improvement iterations prior to intensification, $\text{Limit}$</td>
<td>0.02x$\text{outer_loop}$</td>
</tr>
<tr>
<td>Length of tabu list, $\text{length}$</td>
<td>n/2</td>
</tr>
</tbody>
</table>

**B. Results**

In order to evaluate the performance of our proposed algorithm, we decided to solve some benchmark problems from a library for research on the QAP (QAPLIB) which have been studied and solved by other researchers [31]. For each benchmark problem, the proposed algorithm was executed 20 times with different random seeds.

According to [37], the instances of QAPLIB can be classified into four classes: unstructured (randomly generated) instances, grid-based distance matrix and real-life instances and real-life-like instances. Due to the limitation of the target algorithm that can only solve symmetric instances with zero diagonal values, we only focus on some instances from three classes: unstructured (randomly generated) instances, grid-based distance matrix and real-life instances. Table II presents the instances selected from each class.

<table>
<thead>
<tr>
<th>Class</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>unstructured (randomly generated) instances</td>
</tr>
<tr>
<td>II</td>
<td>grid-based distance matrix</td>
</tr>
<tr>
<td>III</td>
<td>real-life instances</td>
</tr>
</tbody>
</table>

The following tables summarize the average objective function value obtained and the best objective function value obtained for each class. The objective function values of the optimal/best known solutions given in Burckard et al. [31] are also presented for comparison purposes. The heading $\Phi_1$ refers to the percentage deviation between the average objective function value of the solutions obtained and the best known/optimal solution, while the heading $\Phi_2$ refers to the percentage deviation between the best objective function value of the solutions obtained and the best known/optimal solution. The values for $\Phi_1$ and $\Phi_2$ are computed as follows:

$$\Phi_1 = 100 \left( \frac{\text{best known/OP} \text{sol} - \text{average objective function value of algorithm}}{\text{best known/OP} \text{sol}} \right)$$  \hspace{1cm} (8)$$

$$\Phi_2 = 100 \left( \frac{\text{best known/OP} \text{sol} - \text{best objective function value of algorithm}}{\text{best known/OP} \text{sol}} \right)$$  \hspace{1cm} (9)$$

Table III, IV and V summarize the computational results of problem instances in the first class. From Table III, we
notice that the average gaps of the solutions are less than or equal to 0.40%. For each problem instance, the hybrid algorithm is again able to obtain the best known/optimal solutions. The value of $\Phi_1$ is not more than 0.03% for row problem instances (Table IV).

As shown in Table V, for the tai type benchmarks, the performance of the proposed algorithm is still acceptable with values of $\Phi_1$ and $\Phi_2$ are not more than 3.72% and 3.58%, respectively. For larger problem instances (with $n > 20$), the best known/optimal solutions cannot be found. It is likely that with greater number of iterations, the outcome may improve with possibility of obtaining the best known/optimal solutions for some of instances.

The computational results for the second class (grid-based distance matrix) are summarized in Tables VI, VII and VIII. Table VI is a summary of the results for the problem instances. The results indicate that these problem instances do not pose much difficulty for the proposed hybrid algorithm to obtain good solutions as the values of $\Phi_1$ are not more than 0.03%. All the best known/optimal solutions can be obtained within reasonable computation time, with the optimal solution to the largest problem instance, $nug30$, being obtained within 15 minutes.

Tables VII and VIII show the results of testing on $scr$ and $sko$ problem instances. The values of $\Phi_1$ are 0% for $scr$ problem instances, while the maximum value of $\Phi_1$ is only 0.18% for $sko$ problem instances. For $sko49$ and $sko56$, the values of $\Phi_2$ are about 0.1% from the optimal/best known solution. The longest CPU time required to obtain the solution is about 3 hours for $sko56$.

Finally, Table IX and X summarize the results of testing on the third class (real-life instances). Table IX summarizes the computational results for $chr$ problem instances.

<table>
<thead>
<tr>
<th>Benchmark problem</th>
<th>Optimal/Best known solution</th>
<th>Average Solution</th>
<th>Best Solution</th>
<th>$\Phi_1$ (%)</th>
<th>$\Phi_2$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nug12</td>
<td>578</td>
<td>578</td>
<td>578</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>nug14</td>
<td>1014</td>
<td>1014</td>
<td>1014</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>nug15</td>
<td>1150</td>
<td>1150</td>
<td>1150</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>nug20</td>
<td>2570</td>
<td>2570</td>
<td>2570</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>nug21</td>
<td>2438</td>
<td>2438</td>
<td>2438</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>nug22</td>
<td>3596</td>
<td>3596</td>
<td>3596</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>nug24</td>
<td>3488</td>
<td>3488</td>
<td>3488</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>nug25</td>
<td>3744</td>
<td>3744</td>
<td>3744</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>nug27</td>
<td>5234</td>
<td>5234</td>
<td>5234</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>nug28</td>
<td>5166</td>
<td>5166.9</td>
<td>5166</td>
<td>0.02</td>
<td>0.00</td>
</tr>
<tr>
<td>nug30</td>
<td>6124</td>
<td>6124.4</td>
<td>6124</td>
<td>0.01</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark problem</th>
<th>Optimal/Best known solution</th>
<th>Average Solution</th>
<th>Best Solution</th>
<th>$\Phi_1$ (%)</th>
<th>$\Phi_2$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>scr12</td>
<td>31410</td>
<td>31410</td>
<td>31410</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>scr15</td>
<td>51140</td>
<td>51140</td>
<td>51140</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>scr20</td>
<td>110030</td>
<td>110030</td>
<td>110030</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark problem</th>
<th>Optimal/Best known solution</th>
<th>Average Solution</th>
<th>Best Solution</th>
<th>$\Phi_1$ (%)</th>
<th>$\Phi_2$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sko42</td>
<td>15812</td>
<td>15833.8</td>
<td>15812</td>
<td>0.14</td>
<td>0.00</td>
</tr>
<tr>
<td>sko49</td>
<td>23386</td>
<td>23424.5</td>
<td>23410</td>
<td>0.16</td>
<td>0.10</td>
</tr>
<tr>
<td>sko56</td>
<td>34458</td>
<td>34520.4</td>
<td>34494</td>
<td>0.18</td>
<td>0.10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark problem</th>
<th>Optimal/Best known solution</th>
<th>Average Solution</th>
<th>Best Solution</th>
<th>$\Phi_1$ (%)</th>
<th>$\Phi_2$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr12a</td>
<td>9552</td>
<td>9552</td>
<td>9552</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>chr12b</td>
<td>9742</td>
<td>9742</td>
<td>9742</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>chr12c</td>
<td>11156</td>
<td>11156</td>
<td>11156</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>chr15a</td>
<td>99966</td>
<td>99966</td>
<td>99966</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>chr15b</td>
<td>7990</td>
<td>7990</td>
<td>7990</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>chr15c</td>
<td>9504</td>
<td>9504</td>
<td>9504</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>chr18a</td>
<td>11098</td>
<td>11098</td>
<td>11098</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>chr18b</td>
<td>1534</td>
<td>1534</td>
<td>1534</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>chr20a</td>
<td>2192</td>
<td>2224.9</td>
<td>2192</td>
<td>1.50</td>
<td>0.00</td>
</tr>
<tr>
<td>chr20b</td>
<td>2298</td>
<td>2306.7</td>
<td>2298</td>
<td>0.38</td>
<td>0.00</td>
</tr>
<tr>
<td>chr23a</td>
<td>14142</td>
<td>14142</td>
<td>14142</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>chr22a</td>
<td>6156</td>
<td>6181.3</td>
<td>6156</td>
<td>0.41</td>
<td>0.00</td>
</tr>
<tr>
<td>chr22b</td>
<td>6194</td>
<td>6265.2</td>
<td>6194</td>
<td>1.15</td>
<td>0.00</td>
</tr>
<tr>
<td>chr25a</td>
<td>3796</td>
<td>3811</td>
<td>3796</td>
<td>0.40</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark problem</th>
<th>Optimal/Best known solution</th>
<th>Average Solution</th>
<th>Best Solution</th>
<th>$\Phi_1$ (%)</th>
<th>$\Phi_2$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>kra30a</td>
<td>88900</td>
<td>89554.5</td>
<td>88900</td>
<td>0.74</td>
<td>0.00</td>
</tr>
<tr>
<td>kra30b</td>
<td>91420</td>
<td>91420</td>
<td>91420</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>kra32</td>
<td>88700</td>
<td>88700</td>
<td>88700</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
difficulty level in solving the chr problem instances is considered significant [25]. On the whole, the proposed hybrid algorithm is able to find solutions with values of $\Phi$, not exceeding 1.50% from the known optimum. For all problem instances, the best known/optimal solutions are also obtained.

Tables X summarizes the results of testing on kra problem instances. The average gaps of the solutions are less than 0.75%. For each problem instance, the hybrid algorithm is again able to obtain the best known/optimal solutions.

In summary, we observe that the proposed hybrid algorithm is able to obtain very good or optimal solutions to benchmark problem instances drawn from the QAPLIB. The computation time required to do so is also reasonable especially for problem instances with modest size.

V. CONCLUSIONS

In this paper, a hybrid algorithm that combines GRASP, Simulated Annealing and Tabu Search is proposed to solve the QAP. The proposed algorithm for solving the QAP is able to obtain the optimal or best known solutions for problem instances drawn from the QAPLIB.

There are several issues for future research. First, in the proposed hybrid algorithm, the Tabu Search framework has been designed primarily with short term memory. As part of future research work, the possibility of implementing other Tabu Search strategies, such as long term memory and diversification strategy, within the hybrid algorithm will be considered. Second, different types of hybridization with other metaheuristics, such as the genetic algorithm and ant colony optimization algorithm, can also be investigated. Third, the application of the proposed hybrid algorithm to solve other optimization problems is another area of future research, such as Quadratic Semi Assignment Problem (QSAP).

REFERENCES


