Multi-Start Tabu Search and Diversification Strategies for the Quadratic Assignment Problem

Tabitha James[a,1], César Rego[b], and Fred Glover[c]

a Department of Business Information Technology, Pamplin College of Business, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA. tajames@vt.edu

b School of Business Administration, University of Mississippi, University, MS 38677, USA. crego@bus.olemiss.edu

c University of Colorado, Boulder, CO 80309-0419, USA. fred.glover@colorado.edu

Latest Revision: November 21, 2006.

Abstract – The quadratic assignment problem (QAP) is a well known combinatorial optimization problem most commonly used to model the facility-location problem. The widely acknowledged difficulty of the QAP has made it the focus of many metaheuristic solution approaches. In this study, we introduce several multi-start tabu search variants and show the benefit of utilizing strategic diversification within the tabu search framework for the QAP. Computational results for a set of problems obtained from QAPLIB demonstrate the ability of our TS multi-start variants to improve on the classic tabu search approach that is one of the principal and most widely used methods for the QAP. We also show that our new procedures are highly competitive with the best recently introduced methods from the literature, including more complex hybrid approaches that incorporate a classic tabu search method as a subroutine.

Keywords: tabu search, combinatorial optimization, quadratic assignment problem.

[1] Corresponding author.

1. Introduction

The quadratic assignment problem (QAP) is an NP-hard combinatorial optimization problem first introduced by Koopmans and Beckmann [22] to model a facility location problem. In this context, the objective is to find a minimum cost assignment of facilities to locations considering the flow of materials between facilities and the distance between locations. The problem can be formulated as follows:

$$\min_{p \in \Pi} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)}$$

where f is the flow matrix, d is the distance matrix, p is a permutation vector of n indexes (of facilities or locations) mapping a possible assignment of n facilities to n locations, and Π is the set of all n-vector permutations. For each pair of assignments in p the flow between the two facilities is multiplied by the distance between the two locations. The sum of these terms over all pairs gives the total cost assignment for the permutation p. The objective is to find a permutation p^* in Π of minimum total cost.

Although facility location has been the most popular application of the QAP, a great number of other applications have also been encountered in a variety of other domains. Blackboard wiring problem in electronics, arrangement of electronic components in printed circuit boards and in microchips, balancing turbine runners, analysis of chemical reactions in chemistry, machine scheduling in manufacturing, load balancing and task allocation in parallel and distributed computing, statistical data analysis, information retrieval, and transportation are among the better known examples of applications of the QAP in systems engineering [6]. It is also possible to formulate several other well-known combinatorial optimization problems as QAPs, including the traveling salesman problem, the maximum clique problem and the graph-partitioning problem, each of them individually embracing a wide range of other applications in industry, technology and engineering. Featured articles on these application domains and special cases may be found in [3, 9, 16, 21, 31, 32, 33, 34, 42].

Due to its solution complexity and its broad applicability, the QAP has been the subject of extensive research in the realms of both exact solution approaches and metaheuristic approaches. The computational limits of existing technology make exact approaches impractical for all but relatively small problem instances. Metaheuristic approaches have therefore become popular alternatives due to their superior ability to obtain good quality solutions within the limitations of available computing resources.

Metaheuristic approaches applied to the QAP have included artificial neural networks [4], simulated annealing [8, 41], genetic algorithms [7, 39], tabu search [2, 28, 35, 38], ant colony optimization [15, 25, 37], GRASP [23, 30], scatter search [10] and path-relinking [20]. Many variations of these approaches are present in the literature of the QAP, e.g. [1, 11, 12, 13, 14, 19, 24, 26, 27, 29, 40]. In section 2, we survey the current most advanced of these approaches for the QAP. Almost all recently successful methods have involved hybrids of some type. The commonality among all these approaches is the use of a local search method, typically incorporating adaptive memory strategies from tabu search, embedded within the proposed metaheuristic framework. Most of the better approaches explicitly incorporate some variation of a tabu search algorithm developed by Taillard [38]. Taillard's robust tabu search (RTS) algorithm on its own obtains good solutions to the QAP and is very inexpensive in terms of computational time.

2. Review of Leading Approaches

A number of the recent metaheuristic solution techniques that have appeared in the literature have been genetic algorithm (GA) variants coupled with tabu search. Misevicius proposed two GA approaches joined with tabu search for the QAP, a genetic algorithm hybridized with a “ruin and recreate” procedure [26] and an improved hybrid genetic algorithm utilizing a “shift mutation operator” [27]. Both of these approaches are superimposed on a modified version of Taillard’s tabu search which is used to execute the key function of improving the solutions provided by the GA operators. The ruin and recreate procedure uses an operator to randomly perturb the solutions provided by the GA. The tabu search procedure is then applied as an operator to “recreate” solutions provided by the GA operators as well as the “ruined” solutions created by the perturbations [26]. The more advanced hybrid GA version of this approach adds a “shift mutation” operator that further perturbs selected solutions to create greater diversification [27].

Misevicius [28] also introduces a tabu search variant utilizing some of the same concepts of the previous methods but removing the GA operators. This algorithm simply alternatively applies the local search (the modified tabu search procedure of Taillard) to solutions that are periodically subjected to mutations (i.e., perturbations). Taillard’s RTS algorithm was amended to exclude the aspiration criteria, decrease the tabu tenure and simplify the tabu condition. Supplementing this, several diversifying “mutation” processes were incorporated, including a random pairwise exchange procedure, a shift procedure, a dichotomic mutation (exchanging halves of the permutation) and a neighbor exchange mutation (exchanging two adjacent assignments). Misevicius’s approach can be viewed as a restarted tabu search using diversification operators similar to some of those applied in the current study.

Two other hybrid genetic algorithm approaches are given by Drezner [12], [13] which similarly incorporate several tabu search variations within a modified GA framework. Drezner [12] examines the use of a descent heuristic, a simple tabu search, and a new “concentric” tabu search procedure, which identifies and evaluates candidate moves based upon their distance from a “center” solution. Drezner [13] incorporates an extension of the concentric tabu search approach that considers a larger number of permissible moves.

A GRASP implementation utilizing local search is presented by Li, Pardalos, and Resende [23], and a more advanced variant of GRASP enhanced by a TS path relinking strategy is introduced in Oliveira, Pardalos, and Resende [29]. Another path-relinking approach using Taillard’s RTS procedure as an improvement method is introduced and studied in James, Rego, and Glover [20]. Recently, a hybrid metaheuristic approach combining ant colony optimization with a genetic algorithm and local search has been proposed by Tseng and Liang [40].

Due to differences in the QAP test sets chosen for testing, direct comparisons of the techniques outlined above are difficult. The best performing algorithms from the literature typically provide computational results for different test instances from QAPLIB. The hybrid GAs introduced by Misevicius [26], [27] and the tabu search variants [28] produce some of the best solutions for the Taillard test sets but are not run on any other test instances. The hybrid GA due to Drezner [13] produces the best quality solutions for the Skorin-Kapov test suite. No results are presented for the Taillard test instances for Drezner’s algorithms. The GRASP algorithms [23], [29] do not present results for either the Taillard test problems or the Skorin-Kapov test problems, so performance comparisons between the GRASP implementations and the other approaches outlined above are not possible. The path relinking approach [20] and the hybrid metaheuristic approach [40] are both shown to be competitive on the asymmetric Taillard instances. They are also competitive with several of the hybrid GA approaches for the Skorin-Kapov test instances.

While this is not an exhaustive list of approaches for the QAP, the algorithms discussed constitute many of the best performing approaches found in the current literature. As noted, all utilize some variation of a local search procedure, typically either Taillard's RTS procedure or a modification of it. In this study we examine various tabu search strategies using the RTS algorithm as a benchmark to demonstrate the contribution of these strategies. The resulting multi-start tabu search approaches demonstrate that high quality results can be obtained by simple and fast procedures incorporating traditional tabu search intensification and diversification without requiring the complicated designs introduced in the more elaborate "hybrid" metaheuristics. Our approaches are both efficient and easy to implement. Because of this, our improved TS algorithms can also easily be embedded within more complex metaheuristic procedures such as those described above.

3. Tabu Search for the QAP

The hallmark of tabu search is the use of adaptive memory to guide the exploration of the search space. Basic (rudimentary) TS procedures make use of a short term "recency memory" to exclude consideration of moves that lead back to recently visited solutions, together with one or more aspiration criteria that override the tabu status of moves that have suitably attractive properties. More advanced tabu search procedures incorporate additional short term and longer term memory structures, including those based on frequency and on logical analysis. Accompanying these memory structures are intensification and diversification strategies, which respectively focus the search in regions previously found to contain good solutions and drive the search into promising new regions not previously visited. (See [17] for a comprehensive coverage of TS.) In this paper we focus primarily on the simpler TS strategies, utilizing intensification and diversification processes that are straightforward and easy to implement.

The RTS algorithm developed by Taillard [38] provides a core tabu search method that itself embodies simple components and that has been shown to provide high quality solutions to the QAP with a small expenditure of solution time. As in the case of many of the other leading QAP methods, our current study incorporates many of the design features of the RTS algorithm. We also use this method as a benchmark for evaluating the contribution of various multi-start TS procedures we have developed. The remainder of this section describes the basic components of our approaches.

3.1 Neighborhood Specification

The neighborhood most commonly employed in local search algorithms for the QAP, including RTS and the multi-start methods presented in the current study, is the classical 2-exchange (or swap) neighborhood.

To illustrate, consider the following permutation:

$$p(1) = 3 \;\; 12 \;\; 1 \;\; 5 \;\; 8 \;\; 7 \;\; 2 \;\; 9 \;\; 4 \;\; 10 \;\; 6 \;\; 11$$

It is convenient to view each location in the permutation as representing a facility. Entries in the permutation therefore represent the assigned location of each facility. The above permutation therefore represents the assignment of facility 1 to location 3, facility 2 to location 12, and so on. It is straightforward to reverse this encoding.

A move in the 2-exchange neighborhood consists of exchanging (or swapping) two locations. For example, a move denoted by (5,9) results in the following permutation:

$p(2)$ = 3 12 1 9 8 7 2 5 4 10 6 11.

The new solution now assigns facility 4 to location 9 and facility 8 to location 5. The neighborhood thus constitutes the set of all possible moves of this type.

Other encodings and neighborhoods (such as *k*-exchange neighborhoods) have also been considered in the literature, but the computational burden of the larger exchange neighborhoods has limited their use.

In the case of the simple 2-swap neighborhood selected here, the value of each possible new permutation created by a swap move could simply be calculated based on the objective function given in Section 1. However, as the size of the problem grows, such an explicit calculation becomes expensive. One of the key elements of Taillard's RTS is a procedure that quickly ascertains the impact of the 2-exchange moves on a given permutation, thereby saving computational cost, as we examine next.

3.2 Cost Calculation

To expedite the evaluation of a 2-exchange move, the RTS algorithm utilizes a matrix to store the cost associated with each swap that may be executed in the current permutation. These "partial costs" can then be added to the original cost of the permutation to obtain the value associated with the new permutation. In this manner, the costs of possible moves can be quickly evaluated and once a move is chosen, the matrix can be efficiently updated to reflect the costs associated with the newly formed permutation. These partial costs, for symmetrical QAP instances, can be calculated for a move (r,s) on permutation p by:

$$\Delta(p,r,s) = 2\sum_{k \neq r,s} (f_{sk} - f_{rk})(d_{p(s)p(k)} - d_{p(r)p(k)}).$$

Once a move (r,s) is chosen, it is then possible to update the move cost matrix for symmetrical instances by the formula:

$$\Delta(t,u,v) = \Delta(p,u,v) + 2(f_{ru} - f_{rv} + f_{sv} - f_{su})(d_{t(s)t(u)} - d_{t(s)t(v)} + d_{t(r)t(v)} - d_{t(r)t(u)}),$$

where t is the new permutation and u and v differ from r and s. If u or v is the same as r or s, then the first equation can be used to compute the cost. (See [38] for a more detailed discussion of these costs and [5] for a discussion of asymmetrical cases.)

3.3 Tabu List

The tabu list to carry out a short term memory function maintains a record of previously accepted moves by assigning these moves a tabu tenure that denotes the length of time (typically in iterations) during which the elements of a previous move are considered "tabu" and hence a move consisting of such elements is forbidden. In the current study, we adopt the rules for designating an exchange tabu utilized in the RTS algorithm. However, our use of a multi-start component changes the basic structure of the algorithm as will be discussed in later sections. Modifications to the maintenance of the tabu list matrix are performed in the mulit-start algorithms that result in a different structure of the tabu list than in RTS.

To determine tabu status, we maintain a matrix of tabu tenures for each element, starting from a tabu tenure matrix in which all moves are permissible. Once a move is accepted, an updated tabu tenure is assigned to both elements of the exchange and stored in the matrix. This updated tenure results by adding a random number from a defined range to the iteration count, making the associated elements tabu for a specified number of iterations beyond the current iteration. The tabu condition prevents a move from being executed only if both elements of the exchange are currently tabu.

3.4 Aspiration Criteria

An aspiration criterion is a rule that allows the tabu status to be overridden in cases where the forbidden exchange exhibits desirable properties. The aspiration criteria incorporated into all variants developed in this study are the same as those used in the RTS algorithm.

The aspiration criteria utilized in this study require a tabu move to successfully pass through a series of three levels of criteria to ultimately become a permissible exchange. The first level necessitates that an exchange meet one of two criteria. The first determines if the forbidden move results in a global best solution (the solution has the best objective function value of any solution found so far during the search). The second establishes whether or not the tabu tenure of at least one of the two elements of the exchange is less than a predefined ceiling (the iteration minus a defined aspiration value). If the forbidden move meets either requirement then it is marked as potentially admissible and is subject to the second level criterion.

The second level criterion determines if the tabu exchange under consideration is the first forbidden move examined in the current iteration of the algorithm. If the exchange is the first move to override the tabu status for the current working permutation, then the move is permitted. Otherwise, the exchange is subjected to one final criterion.

The third level criterion determines the quality of the exchange in relation to the cost of the previous exchanges examined during the current iteration. This comparison examines the move (or partial) cost rather than the objective function value of the permutation after the exchange. If the cost of the forbidden exchange is better than all of the previous exchanges examined on the current working solution, the move becomes permissible.

3.5 The Traditional Tabu Search Algorithm

The outline of a traditional tabu search algorithm is provided in Figure 1. The outer loop of Figure 1 determines the number of iterations the algorithm is allowed before execution ceases (the stopping condition). The stopping condition applied in a traditional TS, may be based on solution quality, execution time, or iterations. The stopping condition utilized in all variations of the TS algorithms in the current study, including the RTS used for comparisons, is to stop the execution of the algorithm after the global best is not updated for a defined number of iterations. In the original RTS algorithm, the stopping condition was a defined number of iterations regardless of improvement. For this study, the stopping criterion was modified in the original RTS to allow for valid comparisons with the proposed multi-start variants.

```
Loop while (num_failures < max_failures)
    If is tabu but meets all aspiration criteria or is not tabu and best cost so far
        store best exchange that meets all conditions
    End If
    update tabu list
    make exchange on working solution
    If strictly improving
        update best solution
    Else
        increment num_failures
    End If
End Loop
```

Figure 1 - Tabu Search Framework

The main logic of the algorithm begins by setting the working solution to a randomly drawn permutation and calculating the partial cost matrix for this permutation. All possible swaps are considered and the best non-tabu or aspired move is accepted. The chosen move is not necessarily globally improving, but that move is still made on the working solution. If the permutation resulting from the move is globally improving the global best solution is appropriately updated. After an exchange is chosen, the tabu tenure for each element of the accepted exchange is updated. The partial cost matrix for the permutation is also appropriately updated to reflect the exchange made on the old working solution and the algorithm repeats by choosing the next desired exchange on the new working solution.

If the components described in the previous sections are inserted into the skeleton in Figure 1, the resulting algorithm is an RTS with a modified stopping condition. The multi-start variants developed in this study change the structure of this skeleton and will be discussed in depth in the following sections.

4. Multi-Start Tabu Search

The algorithm skeleton provided in Figure 1 assumes that the algorithm is seeded with one randomly generated permutation and operates on that permutation until a stopping criterion is met. A multi-start approach differs from this traditional design by adjusting the search strategy to perturb the standard search path. This may be achieved by running concurrent or sequential independent searches from multiple starting solutions, identifying the best solution from all searches at the end. The perturbation may also be accomplished by strategically restarting a traditional TS at various points during the exploration. The restart may consist of modifying the search trajectory, restarting the search from a different position in the search space or both. In this way, multiple starts of the TS algorithm are undertaken but at strategic times during the search and possibly utilizing information obtained from the prior iterations of the search. The multi-start variants developed in this study use the second strategy.

We explore several methods of restarting the search when an undesirable amount of stagnation has occurred. All of the algorithms developed for this study begin as traditional RTS algorithms as described above. As the search proceeds, however, the number of failed consecutive attempts to update the best solution found are recorded and compared against a threshold value. When this threshold value is exceeded, the variants are restarted. As is shown in Figure 2, the skeleton of the multi-start variants is identical to the traditional TS given in Figure 1 with the exception of an additional parameter and condition block. In all of the multi-start variants, the number of allowable failures is a value drawn from a defined range, the upper limit of this range being less than the maximum failures that defines the stopping criterion in the traditional TS. If the allowable failures threshold is exceeded in the algorithm, a restart is performed that perturbs the search in some manner and the allowable failures parameter is reset. The algorithm continues to iterate, restarting when the threshold condition is violated, until the original stopping condition is met.

Our multi-start approaches can be separated into two categories. The first category contains the variants that perturb the search by modifying some algorithm parameters but continue the exploration from the same working solution. By modifying the parameters when a restart occurs, the permissible exchanges for subsequent iterations of the algorithm are altered, allowing for the possibility that the algorithm proceeds on a different search trajectory than that of the traditional TS. Since the multi-start variants continue from the same working solution they do not alter the position in the search space from which the search continues. The restricted descent tabu search (RDTS) and the tabu tenure modification tabu search (TTMTS) both fall into this category, where the alteration of the tabu list matrix and the tabu tenure parameters are explored. The tabu list matrix contains the current tabu status of the elements that establish whether or not an exchange is forbidden. The tabu tenure parameters designate the range of values from which the tabu tenure for an element is chosen. The value (tabu tenure) drawn from this range determines the length of time an exchange is forbidden. These parameters may also impact the aspiration criteria. As mentioned previously, a solution passes a first level aspiration test if the tabu status of one of the elements of an exchange is less than a defined ceiling. The aspiration value parameter is not modified in the multi-starts.

```
Loop while (num_failures < max_failures)
    If is tabu but meets all aspiration criteria or is not tabu and best cost so far
        store best exchange that meets all conditions
    End If
    update tabu list
    make exchange on working solution
    If strictly improving
        update best solution
    Else
        increment num_failures
        If (num_failures = allowable_failures)
            Perform restart
            Reset allowable_failures
        End If
    End If
End Loop
```
Figure 2 – Multi-Start Tabu Search Framework

The second category contains the random restart tabu search (RRTS), the best solution found tabu search (BSFTS) and the diversified best solution found tabu search (DivTS). In these variants, the parameter modifications are combined with the replacement of the working solution by some other permutation. The new starting solution replaces the current working solution in these algorithms and is either a randomly drawn solution, the global best solution, or a diversified version of the global best solution respectively. While all these variants preserve the global best solution found, they differ with respect to how they use the previous search history to proceed. The restart that occurs in the RRTS variant is identical to starting a new TS with modified tabu tenure range parameters except that the best solution found is still retained for comparison. The other two variants replace the current working solution with a solution, or variation of a solution, that is already known to be favorable, thus utilizing more information from the previous iterations of the search.

The multi-starts change the structure of the algorithm presented in Figure 1 by forcefully perturbing (or restarting) the search in some manner when an undesirable amount of stagnation occurs. Each variant follows the skeleton given in Figure 2 and differs only in the manner of restart applied. The restart that occurs when the allowable failures threshold is reached for each of the five variants will be detailed in the following sections.

4.1 Restricted Descent Approach (RDTS)

The first multi-start variant is the simplest approach. The restart component of this variant modifies only the tabu list matrix parameter. As the search begins to stagnate (the allowable failures condition is met), the current tabu status of all elements is released by resetting the tabu list matrix to once again allow all possible moves from the current working solution. The tabu list is then rebuilt as subsequent moves are made and the algorithm continues to iterate until either the allowable failures condition is met again or the stopping condition is reached. In this manner, a restricted descent from the current working solution occurs at the restart since a greedier move selection is allowed initially as the tabu list matrix is being recreated.

In the RDTS variant, the tabu tenure parameters are defined as in RTS and left constant throughout the entire run of the algorithm. The restart applied in this variant is only in terms of a parameter adjustment, the current working solution is not replaced. The current working solution obtained from the previous iterations of the algorithm is retained. The intention of this variant was to gauge the impact of releasing the tabu list in order to benchmark the more sophisticated modifications.

4.2 Tabu Tenure Parameter Modification Approach (TTMTS)

The TTMTS variant takes the previous approach one step further. As in the previous variant (RDTS), the tabu list is released (cleared). The TTMTS variant then additionally modifies the tabu tenure parameters. The tabu tenure for an exchanged element is drawn from a defined range, added to the current iteration count, and subsequently stored in the tabu list matrix. To obtain the tabu tenure of an element, an upper and lower value is required to determine the range. Both values are dependent on the size of the QAP instance being examined. However, in the RTS algorithm, these values were kept constant and for a particular instance the range did not change over the run of the algorithm.

In the TTMTS variant, when the allowable failures condition is reached, not only is the tabu list released, new upper and lower values to determine the tabu tenure range are chosen. At each restart of the algorithm, therefore, it is possible that the range of tabu tenure values can tighten or loosen. This could result in the tabu status of elements expiring closer together for subsequent iterations of the algorithm. It may also result in a larger divergence in the length of time elements are tabu. The range values may increase or decrease from the previous settings. This means that the length of the tabu status of all exchanged elements may correspondingly increase or decrease for subsequent iterations of the algorithm. This parameter modification changes the set of permissible moves as the algorithm continues from each restart. Similar to the tabu list parameter modification, this may change the search trajectory.

Again, the current working solution in this variant is not replaced, the search continues from the same working solution obtained from the iteration previous to the restart. As with the last version, the search is restarted only in the sense of making parameter adjustments and still retains a small part of the search history by leaving untouched the current working solution.

4.3 Random Restart Approach (RRTS)

The random restart variant (RRTS) restarts the search from a randomly drawn permutation with new tabu tenure parameters and a released tabu list. It restarts when the algorithm stagnates and saves only the global best solution. This approach is similar to running a traditional TS algorithm, with new tabu tenure parameters for each search phase, which is run several times for a shorter number of iterations. The algorithm continues from a better (global best) solution than a completely fresh run of a TS algorithm.

As in the TTMTS variant, the RRTS algorithm restarts under the new tabu tenure parameters and a released tabu list, but does not restart from the current working solution. Rather, the current working solution is replaced with a randomly drawn permutation generated in the same manner as the initial working solution. The only information saved from the previous iterations of the search is the global best solution. However, this previous search information may impact the aspiration criteria since a permutation created from a move may be less likely to improve upon the global best solution.

4.4 Best Solution Found Restart Approach (BSFTS)

As in the previous two variants, the same modifications to the tabu list matrix and the tabu tenure parameters are made in the BSFTS variant. The difference in this variant is that the working solution from which the algorithm continues after the restart is the best solution found up to that point in the search. This allows for a restricted descent from the global best solution under new tabu tenure parameters. In this case, the idea is to capitalize on the best information currently available and perturb the search from this region with the adjustment of the tabu tenure parameters. This intensifies the search, in a simple manner, as it forces the search to restart from an already promising region.

4.5 Diversified Best Solution Found Restart Approach (DivTS)

The diversified best solution found (DivTS) variant also releases the tabu list and modifies the tabu tenure parameters in the same manner as above. The DivTS variant again differs in the replacement of the current working solution at the restart. This variant replaces the current working solution with a strategically diversified permutation created from the global best solution found up to the restart condition being met.

This new working solution may differ greatly from the current global best solution and the current working solution, but it is not randomly generated as in the RRTS variant. The DivTS approach forcefully diversifies the search, but in a more tactical manner than a random restart. The diversification procedure used provides a certain level of assured variability in the solution from which the algorithm is restarted. The method used to create the new working solution from the current BSF is described in the following subsection.

Diversification Procedure

The diversification procedure used in DivTS is suggested by Glover [18]. The pseudocode for this procedure is given in Figure 3. The method generates new starting solutions from a randomly generated seed solution in the manner illustrated below.

Given a randomly generated permutation such as *s*, where *s*(1)…s(*n*) are the locations to which facilities 1…*n* are assigned:

$$s = (8,\ 1,\ 5,\ 10,\ 9,\ 3,\ 7,\ 2,\ 12,\ 11,\ 6,\ 4)$$

A step is defined that determines the increment used to step through the elements of the permutation. The step is also used to initialize the starting position (the start variable in Figure 3). For example, if step = 3, then through the first pass of the inner loop, start = 3, which results in the partial solution:

$$ss = (\mathbf{5},\ \mathbf{3},\ \mathbf{12},\ \mathbf{4})$$

The starting position is then readjusted to start = 2, generating in the next pass of the inner loop:

$$ss = (5,\ 3,\ 12,\ 4,\ \mathbf{1},\ \mathbf{9},\ \mathbf{2},\ \mathbf{6})$$

This process is continued until start = 1, in which case a full starting solution is generated:

$$ss = (5,\ 3,\ 12,\ 4,\ 1,\ 9,\ 2,\ 6,\ \mathbf{8},\ \mathbf{10},\ \mathbf{7},\ \mathbf{11})$$

```
position = 1
For start = step to 1 (decrement start by 1)
    For j = start to n_var (increment j by step)
        starting_solution (position) = seed_solution(j)
        position ++  (increment position by 1)
    End For
End For
```

Figure 3 - Pseudocode for the Diversification Method

This method can be used to generate 1 to *n* solutions from any given permutation, where *n* is the number of variables in the problem instance. The first solution produced is simply the original seed solution. In the DivTS variant, the step size starts at 1 and is increased by 1 at each restart until it equals the problem size and then if necessary is reset to 1. Therefore, at the first restart of DivTS, the current working solution is replaced with the global best solution and at each subsequent restart a diversified version of the global best solution is utilized.

5. Computational Results and Discussion

To compare the mulit-start tabu search variations, a set of 31 test problems obtained from QAPLIB were used. All algorithms were written in C and run on a single Intel Itanium processor (1.3 GHz). The machine used was an SGI Altix running Linux and the Intel compilers. Each algorithm was run 10 times starting from 10 different randomly generated seed solutions (initial working solutions). The parameters for each variation are shown in Table 1.

To provide initial comparisons between the variants, and between the variants and RTS, the stopping condition was set to 5000*n (n being the problem size or number of facilities/locations for the problem) which provided quick run times (less than 15 minutes for all problem sizes). The robust tabu search procedure (RTS) was written identically to Taillard's algorithm with the exception of this stopping criterion. The stopping criterion was changed from a fixed number of iterations to a maximum number of failures rule to provide valid comparisons between RTS and the multi-start variants.

Table 1- TS Variant Parameter Settings

Parameter	**RTS**	**RDTS**	**TTMTS**	**RRTS**	**BSFTS**	**DivTS**	**DivTS**
Maximum Failures (Stopping Criterion)	5000*n	5000*n	5000*n	5000*n	5000*n	5000*n	50000*n
Allowable Failures Lower Limit (Restart Parameter)	n/a	(5000*n) /1000	(5000*n) /1000	(5000*n) /1000	(5000*n) /1000	(5000*n) /1000	(50000*n) /1000
Allowable Failures Upper Limit (Restart Parameter)	n/a	(5000*n) /10	(5000*n) /10	(5000*n) /10	(5000*n) /10	(5000*n) /10	(50000*n) /10
Tabu Tenure Lower Limit	9*n/10 (static)	9*n/10 (static)	1*n/10 (variable)	1*n/10 (variable)	1*n/10 (variable)	1*n/10 (variable)	1*n/10 (variable)
Tabu Tenure Upper Limit	11*n/10 (static)	11*n/10 (static)	11*n/10 (variable)	11*n/10 (variable)	11*n/10 (variable)	11*n/10 (variable)	11*n/10 (variable)
Aspiration Value	n*n*2	n*n*2	n*n*2	n*n*2	n*n*2	n*n*2	n*n*2

The allowable failure limits define a range from which the number of iterations, in which no improving move is found before a restart occurs, is chosen. An initial number of iterations is chosen from this range at the beginning of the algorithm and at each restart a new value is chosen. The upper limit of this range is set to be less than the maximum failures parameter.

The tabu tenure range parameters used by the RTS algorithm are 9*n/10 and 11*n/10. The initial values of the lower and upper tabu tenure range parameters for all variants were also set to these values. All multi-start variants, with the exception of the restricted decent (RDTS) variant, modified these parameter values at restart. The new tabu tenure range parameters, upon restart only, were drawn from the range of values given in Table 1. The upper and lower tabu tenure values are chosen randomly from this range with the only constraint being that the upper value must be larger than the lower value. The aspiration value employed in RTS was used for all variants.

The last column in Table 1 gives the parameter settings for the longer run of the diversified BSF variant (DivTS). The maximum number of failures before stopping was increased to 50000*n and the additional parameters that depended on this value were also modified as is shown in Table 1. Computational testing showed these parameter choices provided desirable results, although further experimentation could possibly provide more insight.

Table 2 provides a comparison of all the variants as well as the RTS algorithm. The average percent deviation (APD) from the best known solution is given for each algorithm as well as the number of times the best known solution was found out of the 10 runs (if it was found) and the average time to completion. All times are in minutes. The best overall average deviation for each test problem is bolded. The shaded cells denote that the variant did as well or better than the RTS algorithm.

Table 2 - Comparative Analysis between TS variants and RTS

		RTS	#		RDTS	#		TTMTS	#		RRTS	#		BSFTS	#		DivTS	#	
Problem	BKS	APD	BSF	Time	APD	BSF	Time	APD	BSF	Time	APD	BSF	Time	APD	BSF	Time	APD	BSF	Time
Sko42	15812	0.003	(9)	0.31	**0.000**	(10)	0.27	**0.000**	(10)	0.28	**0.000**	(10)	0.30	**0.000**	(10)	0.29	**0.000**	(10)	0.29
Sko49	23386	0.079	(1)	0.60	0.068	(1)	0.67	**0.039**	(3)	0.97	0.056	(1)	0.75	0.048	(3)	0.63	0.043	(3)	0.75
Sko56	34458	0.037	(2)	1.59	0.048	(1)	1.38	0.017	(3)	1.42	**0.014**	(3)	1.37	0.031	(2)	1.47	0.016	(3)	1.51
Sko64	48498	0.039		2.66	0.017	(1)	2.69	0.017	(3)	2.82	0.022	(2)	2.13	0.020	(4)	2.53	**0.008**	(5)	2.19
Sko72	66256	0.104		3.26	0.090		3.87	0.068		3.82	**0.065**		4.86	0.072	(1)	4.12	0.087		3.76
Sko81	90998	0.083		4.91	0.087		5.26	0.081		5.82	0.063		7.42	0.113		4.61	**0.057**		5.89
Sko90	115534	**0.088**		7.85	0.116		9.53	0.101		7.92	0.107		7.34	0.155		6.54	0.099		8.96
Sko100a	152002	0.097		9.87	0.107		9.65	0.087		12.98	**0.077**		12.10	0.080		11.14	0.084		13.07
Sko100b	153890	0.083		10.66	0.097		9.89	0.049		12.03	0.067		11.03	0.090		11.20	**0.043**		13.83
Sko100c	147862	0.053		9.68	0.048		10.76	0.032		12.14	0.030		11.24	0.039		11.35	**0.029**		10.83
Sko100d	149576	0.088		10.14	0.097		11.01	**0.073**		12.14	0.081		12.13	0.106		10.58	0.075		12.95
Sko100e	149150	0.067		9.69	0.060		9.46	0.035		11.00	0.061		8.77	0.066		10.26	**0.034**		10.69
Sko100f	149036	**0.085**		11.17	0.107		12.32	0.091		14.33	0.099		10.95	0.141		11.07	0.102		14.08
Tai20a	703482	0.030	(9)	0.24	**0.000**	(10)	0.31	**0.000**	(10)	0.25	0.030	(9)	0.27	0.034	(9)	0.28	0.030	(9)	0.24
Tai25a	1167256	0.047	(9)	0.48	**0.000**	(10)	0.49	0.037	(9)	0.50	0.138	(6)	0.58	0.104	(7)	0.53	0.047	(9)	0.61
Tai30a	1818146	0.205	(1)	0.29	0.117	(2)	0.22	0.184	(3)	0.27	0.186	(5)	0.29	**0.090**	(6)	0.35	0.133	(4)	0.29
Tai35a	2422002	0.619		0.20	0.412	(2)	0.22	0.345	(2)	0.24	**0.292**	(2)	0.27	0.402	(3)	0.21	0.378	(1)	0.21
Tai40a	3139370	0.664		0.31	0.618		0.44	0.544		0.39	0.539		0.39	**0.473**		0.37	0.503		0.36
Tai50a	4938796	1.093		0.70	1.124		1.07	**0.804**		1.31	0.893		1.42	0.920		1.02	0.805		0.96
Tai60a	7205962	1.159		1.83	1.119		2.02	1.028		2.05	**0.951**		2.42	0.959		2.10	1.018		2.34
Tai80a	13515450	1.207		4.51	1.167		4.45	0.979		5.55	0.910		6.97	0.968		6.20	**0.898**		5.65
Tai100a	21059006	1.219		9.94	1.105		10.87	**0.882**		12.08	0.932		11.47	0.925		10.98	0.908		13.14
Tai20b	122455319	**0.000**	(10)	0.20	**0.000**	(10)	0.20	**0.000**	(10)	0.20	**0.000**	(10)	0.20	**0.000**	(10)	0.20	**0.000**	(10)	0.20
Tai25b	344355646	**0.000**	(10)	0.43	**0.000**	(10)	0.43	**0.000**	(10)	0.58	**0.000**	(10)	0.49	**0.000**	(10)	0.52	**0.000**	(10)	0.56
Tai30b	637117113	**0.001**	(6)	0.48	**0.001**	(8)	0.36	**0.001**	(8)	0.36	0.002	(8)	0.28	0.027	(6)	0.50	0.013	(8)	0.42
Tai35b	283315445	**0.000**	(10)	0.16	0.019	(9)	0.20	0.029	(8)	0.20	0.028	(8)	0.24	0.044	(7)	0.18	0.055	(6)	0.16
Tai40b	637250948	**0.000**	(10)	0.25	**0.000**	(10)	0.29	0.001	(9)	0.25	**0.000**	(10)	0.26	**0.000**	(10)	0.24	0.005	(9)	0.23
Tai50b	458821517	**0.037**	(1)	1.01	0.090		1.00	0.101		1.01	0.100		0.97	0.048	(1)	0.91	0.125	(1)	0.90
Tai60b	608215054	**0.013**	(2)	2.34	0.072		2.39	0.116		1.71	0.144		2.07	0.107		2.00	0.139		1.92
Tai80b	818415043	**0.064**		6.26	0.306		5.11	0.296		5.25	0.477		4.84	0.364		4.95	0.406		6.07
Tai100b	1185996137	**0.070**		15.42	0.233		12.55	0.320		12.76	0.396		11.91	0.342		11.66	0.444		9.76

The TS variants developed in this study can be divided into two groups. The first category contains the RDTS and TTMTS variants that manipulate the allowable moves and change the trajectory of the search but continue from the current working solution without replacing it. In these two algorithms there is simply a manipulation of the tabu parameters to perturb the search. The second category is composed of the RRTS, BSFTS, and DivTS variants that replace the current working solution when the restart occurs along with the manipulation of the tabu parameters.

From the first category of variants, TTMTS performs better than RDTS. TTMTS obtains better APDs (average percent deviations from the best known solution) than RDTS for 18 out of the 31 test problems and ties RDTS on 6 instances. In comparison to the RTS algorithm, both the RDTS algorithm and the TTMTS algorithm perform quite well by simply perturbing the set of allowable moves. The RDTS algorithm beats or ties the APD of the RTS algorithm on 18 out of 31 test instances. TTMTS shows even better performance, beating or tying the APD obtained by the RTS algorithm on 23 out of 31 test instances. TTMTS also obtains the overall best APDs over all variants from both categories, as well as RTS, for 4 of the test instances. RDTS obtains the best overall APD for the Tai25a instance.

Of the three multi-start variants that form the second category (RRTS, BSFTS, and DivTS), DivTS shows the best performance. DivTS provides the best APDs of the 3 variants on 12 of the 31 test instances and ties on 4. RRTS is the second best of the variants from the second category, obtaining 8 of the best APDs and 5 ties. BSFTS beats DivTS and RRTS on 6 test instances and ties on 4. All three variants perform very well against the RTS algorithm. DivTS and RRTS obtain better APDs than RTS on approximately 70% of the test instances. BSFTS also obtains better solution quality than RTS on over half the test instances. Comparing only the two best performing multi-starts from this second category, RRTS and DivTS, DivTS is clearly the overall winner obtaining the best APDs on 16 of the 31 test instances compared to 11 for RRTS. DivTS also obtains the best overall APD over all variants, including the RTS algorithm, for 6 of the test instances. BSFTS restarts the search from a promising region and the results indicate an improvement of this approach over RTS. However, the DivTS and RRTS variants obtain the best results which illustrate the benefit of diversification to the search strategy. The improvement of DivTS over the RRTS version demonstrates the importance of using strategic diversification strategies as opposed to those that rely on randomization.

As can be seen in Table 2, all variants performed significantly better than RTS on the Skorin-Kapov problems (Sko*) and the symmetric Taillard instances (Tai*a). For the shorter runs provided in Table 2, the RTS algorithm outperforms all of the variants on the asymmetric instances (Tai*b). The restart parameter (allowable failures) in the variants is dependent upon the maximum failures parameter. As the maximum failures increases, the time to a restart also increases, which allows the base tabu search to operate on a solution for longer without perturbation which improves the asymmetric results. The results provided in Table 3, where the maximum failures parameter (stopping condition) is increased, demonstrate this sensitivity to the restart parameter for the asymmetric test instances.

It is of interest to note that that both the longer run DivTS algorithm and the ETS algorithms that characteristically diversify the search by perturbing the working solution, perform very well on the Tai*b test set. All of the competitive algorithms for the Tai*b test set employ some type of diversification, but our study demonstrates that the timing of these perturbations can dramatically impact the results obtained for this asymmetric problem set. As can be seen in Table 3, the DivTS variant obtains very good results on the Tai*b test set when the time between restarts is extended.

Therefore, the realized benefit seems to depend more on the sensitivity to the allowable failures parameter (or the scheduling of the restart) in this test set than it does in the other test sets.

As can be seen in Table 5, given longer run times, APDs of 0.000 for all but the two largest problems in the Tai*b test set are relatively easy to obtain for all competitive algorithms regardless of the algorithmic design. The symmetric instances, Sko* and Tai*a, prove to be much more difficult for most algorithms. Considering only the symmetric instances in Table 2, the DivTS algorithm outperforms all of the other variants. RDTS obtains 1 of the overall best APDs among all the variants for the symmetric instances. TTMTS and RRTS both obtain 5 of the best APDs each for these 22 test instances. BSFTS obtains 2 of the best overall APDs and DivTS obtains 8 of the best overall APDs. Given DivTS's higher performance on the symmetric instances, where obtaining high quality solutions is more difficult, DivTS was run for longer times and those results are shown in Table 3.

Table 3 presents the long run results for the DivTS algorithm. In these runs the maximum failures parameter was increased to 50000*n. The results presented in Table 3 show that DivTS obtains high quality results for all test instances in relatively short run times. The algorithm obtains APDs of 0.05 or below for all but 6 of the 31 test problems. The best known solution is found 154 out of 310 times (approximately half the total runs). All but one of the average run times are less than 2 hours, with anything under 90 facilities/locations running in less than an hour.

As previously mentioned, the results provided in Table 3 for the asymmetric test instances (Tai*b) demonstrate the sensitivity to the restart parameter for this set of test problems. While the short runs of DivTS in Table 2 provided inferior results for these asymmetric instances, the longer run results are very good. As can be seen in Table 3, the best known solution was obtained 100% of the time for 6 of the 9 test instances. For the remaining 3 instances the APD was under 0.06. For Tai80b, the BKS was found 8 out of the 10 runs with the APD for this test instance being 0.000. Table 3 illustrates that increasing the maximum failures parameter (stopping condition) for the DivTS variant provides results, for the Tai*b instances, that exceed the solution quality of some of the best tabu search algorithms from the literature (ETS) for this asymmetric test set. Although the ETS algorithms provided a new best known solution for one of these instances, it is shown in Table 5 that they do not perform as well as the longer runs of the DivTS algorithm on these asymmetric instances.

Table 3 - Longer Run Results for DivTS

Problem	BKS	DivTS APD	# BSF	Time	Problem	BKS	DivTS APD	# BSF	Time	Problem	BKS	DivTS APD	# BSF	Time
Sko42	15812	0.000	(10)	3.98	Tai20a	703482	0.000	(10)	0.24	Tai20b	122455319	0.000	(10)	0.23
Sko49	23386	0.008	(7)	9.61	Tai25a	1167256	0.000	(10)	0.56	Tai25b	344355646	0.000	(10)	0.46
Sko56	34458	0.002	(8)	13.16	Tai30a	1818146	0.000	(10)	1.31	Tai30b	637117113	0.000	(10)	1.31
Sko64	48498	0.000	(10)	22.03	Tai35a	2422002	0.000	(10)	4.44	Tai35b	283315445	0.000	(10)	2.39
Sko72	66256	0.006	(2)	37.98	Tai40a	3139370	0.222	(1)	5.16	Tai40b	637250948	0.000	(10)	3.18
Sko81	90998	0.016	(2)	56.36	Tai50a	4938796	0.725		10.23	Tai50b	458821517	0.000	(10)	8.82
Sko90	115534	0.026		89.60	Tai60a	7205962	0.718		25.69	Tai60b	608215054	0.000	(8)	17.08
Sko100a	152002	0.027		129.22	Tai80a	13515450	0.753		52.74	Tai80b	818415043	0.006		58.24
Sko100b	153890	0.008	(2)	106.55	Tai100a	21059006	0.825		142.06	Tai100b	1185996137	0.056		118.91
Sko100c	147862	0.006	(2)	126.69										
Sko100d	149576	0.027	(1)	123.45										
Sko100e	149150	0.009	(1)	108.84										
Sko100f	149036	0.023		110.28										

Tables 4 and 5 give comparisons of the DivTS algorithm to the following additional methods from the current literature:

- An Ant Colony Optimization/Genetic Algorithm/Local Search Hybrid – ACO/GA/LS [40]
- A Genetic Algorithm Hybrid with a Strict Descent Operator - GA/SD [12]
- A Genetic Algorithm Hybrid with a Simple Tabu Search Operator - GA/S-TS [12]
- A Genetic Algorithm Hybrid with Concentric Tabu Search Operator - GA/C-TS [12]
- A Genetic Algorithm Hybrid with an Improved Concentric Tabu Search Operator - GA/IC-TS [13]
- Three Tabu Search variants (ETS-1, ETS-2, and ETS-3) [28]
- Two Genetic Algorithm Hybrids with a Tabu Search - GA/TS and GA/TS-I [26], [27]

Tables 4 and 5 provide comparisons of the longer run of the DivTS multi-start variant with some of the best approaches from the literature. The DivTS algorithm is highly competitive with these algorithms both in terms of solution quality and computational time. However, true comparisons are not possible due to the use of different hardware and the lack of full result sets for all algorithms on all test problems. While these comparisons illustrate the strength and competitiveness of the DivTS algorithm with some of the best performing approaches, comprehensive benchmarks were not possible since we were unable to run all algorithms on the same test problems, using the same hardware.

The reported run times over the set of algorithms used for comparisons in Tables 4 and 5 vary dramatically. The longer run of the DivTS algorithm was purposefully set to be comparable and to not exceed the run times of all approaches. Runtime comparisons are made only to illustrate that the algorithms developed in this study are within acceptable ranges. ACO/GA/LS was run on a 2.0 GHz Pentium Intel processor. The clock speed on this processor is faster than the 1.3 GHz Intel Itanium processor used in our study. Straightforward hardware comparisons cannot be made since the processor used in the current study is an Itanium processor (which uses a different instruction set) and the compilers used were different. Benchmarks obtained from SPEC [36] suggest that a 2.0 GHz Pentium processor is slightly slower than a 1.3 GHz Itanium processor. Without knowledge of the exact hardware configurations of the Pentium 2.0 GHz machine, a true performance comparison cannot be made but it may be assumed that these two machines are comparable. GA/SD, GA/S-TS, GA/C-TS, and GA/IC-TS are run on a 600 MHz Pentium III. According to the SPEC benchmarks, the processor used in the current study would be approximately four times as fast as the one used for these GA Hybrids. Therefore, run on a more comparable platform, the runtimes for these algorithms may be much more competitive. Specific hardware or compiler information was not reported for the ETS-*, GA/TS, and GA/TS-I algorithms. The only information provided was that the processor was an x86 Family 6 processor. This can be assumed to be slower than the architecture used in the current study, but any further comparisons are not possible.

Table 4 – Longer Run DivTS Comparisons with Literature for Skorin-Kapov Test Problems

Problem	BKS	DivTS APD	DivTS Time	ACO/GA/LS APD	ACO/GA/LS Time	GA/SD APD	GA/SD Time	GA/S-TS APD	GA/S-TS Time	GA/C-TS APD	GA/C-TS Time	GA/IC-TS APD	GA/IC-TS Time
Sko42	15812	0.000	3.98	0.000	0.71	0.014	0.16	0.001	0.30	0.000	1.15		
Sko49	23386	0.008	9.61	0.056	7.58	0.107	0.28	0.062	0.48	0.009	2.13		
Sko56	34458	0.002	13.16	0.012	9.07	0.054	0.42	0.007	0.72	0.001	3.24	0.000	13.60
Sko64	48498	0.000	22.03	0.004	17.35	0.051	0.73	0.019	1.23	0.000	5.85	0.000	26.18
Sko72	66256	0.006	37.98	0.018	70.83	0.112	0.93	0.056	1.45	0.014	8.36	0.000	38.32
Sko81	90998	0.016	56.36	0.025	112.33	0.087	1.44	0.058	2.18	0.014	13.30	0.003	63.07
Sko90	115534	0.026	89.60	0.042	92.07	0.139	2.31	0.073	3.51	0.011	22.35	0.001	102.25
Sko100a	152002	0.027	129.22	0.021	171.00	0.114	3.42	0.070	5.11	0.018	33.55	0.002	177.56
Sko100b	153890	0.008	106.55	0.012	192.44	0.096	3.47	0.042	5.11	0.011	34.05	0.000	170.18
Sko100c	147862	0.006	126.69	0.005	220.57	0.075	3.22	0.045	4.69	0.003	33.80	0.001	158.38
Sko100d	149576	0.027	123.45	0.029	209.21	0.137	3.45	0.084	5.15	0.049	33.90	0.000	164.22
Sko100e	149150	0.009	108.84	0.002	208.08	0.071	3.31	0.028	4.70	0.002	30.67	0.000	169.61
Sko100f	149036	0.023	110.28	0.034	210.86	0.148	3.55	0.110	5.25	0.032	35.74	0.003	174.55

Table 5 – Longer Run DivTS Comparisons with Literature for Taillard Test Problems

Problem	BKS	DivTS APD	DivTS Time	ETS-1 APD	ETS-2 APD	ETS-3 APD	ETS Time	ACO/GA/LS APD	ACO/GA/LS Time	GA/TS APD	GA/TS-I APD	Time
Tai20a	703482	0.000	0.24	0.000	0.000	0.000	0.03			0.061	0.000	0.04
Tai25a	1167256	0.000	0.56	0.037	0.000	0.015	0.08			0.088	0.000	0.10
Tai30a	1818146	0.000	1.31	0.003	0.041	0.000	0.21	0.341	1.41	0.019	0.000	0.27
Tai35a	2422002	0.000	4.44	0.000	0.000	0.000	3.67	0.487	3.54	0.126	0.000	0.60
Tai40a	3139370	0.222	5.16	0.167	0.130	0.173	28.33	0.593	13.05	0.338	0.209	1.42
Tai50a	4938796	0.725	10.23	0.322	0.354	0.388	116.67	0.901	29.65	0.567	0.424	5.00
Tai60a	7205962	0.718	25.69	0.570	0.603	0.677	116.67	1.068	58.49	0.590	0.547	12.00
Tai80a	13515450	0.753	52.74	0.321	0.390	0.405	200.00	1.178	152.20	0.271	0.320	53.33
Tai100a	21059006	0.825	142.06	0.367	0.371	0.441	666.67	1.115	335.62	0.296	0.259	200.00
Tai20b	122455319	0.000	0.23	0.000	0.000	0.000	0.01			0.000	0.000	0.01
Tai25b	344355646	0.000	0.46	0.000	0.000	0.000	0.02			0.007	0.000	0.02
Tai30b	637117113	0.000	1.31	0.000	0.000	0.000	0.07	0.000	0.25	0.000	0.000	0.03
Tai35b	283315445	0.000	2.39	0.000	0.019	0.000	0.23	0.000	0.32	0.059	0.000	0.05
Tai40b	637250948	0.000	3.18	0.000	0.000	0.000	0.23	0.000	0.59	0.000	0.000	0.12
Tai50b	458821517	0.000	8.82	0.000	0.003	0.000	4.50	0.000	2.89	0.002	0.000	0.33
Tai60b	608215054	0.000	17.08	0.000	0.001	0.003	22.50	0.000	2.83	0.000	0.000	0.67
Tai80b	818415043	0.006	58.24	0.008	0.036	0.016	116.67	0.000	60.29	0.003	0.000	2.50
Tai100b	1185996137	0.056	118.91	0.072	0.123	0.034	333.33	0.010	698.87	0.014	0.000	7.30

The DivTS algorithm outperforms or ties the hybrid ACO/GA/LS in all but 5 cases for the 27 test instances where performance could be compared. In most cases, especially for the larger problem instances, the average run times are also substantially lower.

For the Skorin-Kapov test instances, DivTS outperforms Drezner's GA hybrids with the simple TS operator (GA/S-TS) and the strict descent operator (GA/SD) in all cases, although the run times for DivTS are considerably longer. The original GA hybridized with the concentric tabu search (GA/C-TS) wins 6 out of the 13 problems. This algorithm runs slightly longer than the first two hybridized GAs but still shorter than DivTS. The hybrid method that enhances a GA with an improved concentric tabu search (GA/IC-TS) outperforms the DivTS algorithm in almost all cases with slightly longer run times.

DivTS meets or exceeds the solution quality of the ETS algorithms for all but the larger instances of Taillard's symmetric instances. ETS-3 also slightly edges out DivTS on Tai100b. However, the run times for the ETS algorithms are quite significantly longer than the run times for DivTS. Misevicius's original hybrid GA (GA/TS) again performs better on the larger symmetric instances from the Tai*a test set and on the Tai100b asymmetric instance. The improved version (GA/TS-I) also performs better than DivTS on the same test problems and additionally on Tai80b. The run times for the symmetric instances are comparable to those for DivTS and shorter for the asymmetric instances.

Tseng and Liang (ACO/GA/LS) provide the only study that reports results on both the Sko* and Tai* test instances, omitting only a couple of the smaller problems. The DivTS algorithm reports overall better results than this more complex hybrid in much shorter runtimes. Since this is also the most comparable hardware, the results indicate the ability of the multi-start approach to provide superior performance by utilizing more strategic operators. All of the hybrid GA approaches, as well as the other TS algorithms, report results for only either the Tai* test problems or the Sko* test problems. Drezner (GA/SD, GA/S-TS, GA/C-TS, and GA/IC-TS) reports results for the Sko* problems. Misevicius (ETS-* and GA/TS-*) reports results for the Tai*a and Tai*b test suites. The Sko test set contains structured, symmetric problems only. The Tai*a test set contains unstructured, symmetric problems and the Tai*b problems are unstructured, asymmetric instances. The variation in the nature of the test instances may impact the performance of an algorithm on certain problem types. For example, the asymmetric Tai*b instances appear to be more sensitive to the restart threshold parameter in our study. Without a full result set for all algorithms, it is difficult to determine which, if any, algorithm is superior to all of the others over a full set of QAP instances. The results presented for the longer run of DivTS illustrate that it performs well on all problem sets and is competitive with the algorithms that have been shown to work well on one test set or the other. This illustrates the advantage of the concepts demonstrated in the current study.

6. Conclusions

In this study several multi-start tabu search variants for the quadratic assignment problem were introduced. The algorithms were shown to improve upon the robust tabu search algorithm that is commonly used as a subprocedure in metaheuristic approaches for the QAP. The results demonstrate that merely modifying the tabu parameters, which influences the trajectory of the search by altering the set of allowable moves, can provide improvements in solution quality. The benefits obtained from applying simple intensification and strategic diversification to the search is also illustrated. The success of the diversified best solution found variant (DivTS) shows the benefit of applying strategic diversification rather than relying on randomization. The exploitation of the simple adaptive memory techniques used in the multi-start variants developed indicate the power of using search information and strategic operators to guide the exploration. Future work includes the development and use of more sophisticated intensification and diversification approaches within this multi-start framework.

The results illustrate that high quality results can be obtained from approaches that capitalize on the strategic use of what has already been learned. The multi-start algorithms examined in this study are simple modifications to a traditional tabu search. As such, they execute quickly and are programmatically straightforward. The results are shown to be highly competitive with more complicated hybrid metaheuristic approaches. The success of the multi-start algorithms in comparison to more complex approaches demonstrate the promise of further advancement by applying greater use of intelligently designed strategies. Additionally, the execution speed and programmatic simplicity make them ideal candidates to use in conjunctions with more sophisticated methods such as path relinking or scatter search. This is also a planned area of future research.

References

[1] Ahuja, R.K., Orlin, J.B. and Tiwari, A. (2000). "A descent genetic algorithm for the quadratic assignment problem," Computers and Operations Research, 27:917-934.

[2] Battiti, R. and Tecchiolli, G. (1994). "The reactive tabu search," ORSA Journal on Computing, 6:26–140.

[3] Bhaba R.S, Wilbert E.W. and Gary L.H. (1998). "Locating sets of identical machines in a linear layout," Annals of Operations Research, 77(0):183-207.

[4] Bousono-Calzon C. and Manning M.R.W. (1995). "The hopfield neural network applied to the quadratic assignment problem," Neural Computing and Applications, 3(2):64-72.

[5] Burkard, R.E., and Rendl, F. (1984). "A thermodynamically motivated simulation procedure for combinatorial optimization problems," European Journal of Operational Research, 17, 169-174.

[6] Cela, E. (1998). *The Quadratic Assignment Problem: Theory and Algorithms*, Kluwer Academic Publishers.

[7] Cohoon, J.P., and Paris, W.D. (1987). "Genetic placement," *IEEE Transactions on Computer-Aided Design* 6, 956-964.

[8] Connolly D.T. (1990). "An improved annealing scheme for the QAP," European Journal of Operational Research, 46:93-100.

[9] Coy, S.P., Golden, B.L., Runger, G.C., and Wasil, E.A. (1998). "See the forest before the trees: fine-tuned learning and its application to the traveling salesman problem," IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans, 28(4), 454-464.

[10] Cung, V-D., T. Mautor, P. Michelon, A. Tavares (1996). “Scatter search for the quadratic assignment problem,” Proceedings of the IEEE International Conference on Evolutionary Computation, 165–169.

[11] Dorigo, M.., Maniezzo, V., and Colorni, A. (1996). “Ant system: optimization by a colony of cooperating agents,” IEEE Transactions on Systems, Man and Cybernetics, Part B, 26(1): 29-41.

[12] Drezner, Z. (2003). “A new genetic algorithm for the quadratic assignment problem,” INFORMS Journal on Computing, 15(3), 320-330.

[13] Drezner, Z. (2005). “The extended concentric tabu for the quadratic assignment problem”, European Journal of Operational Research, 160, 416-422.

[14] Fleurent, C. and. Ferland J.A. (1994). “Genetic hybrids for the quadratic assignment problem.” In P. Pardalos and H. Wolkowicz, (Eds.), Quadratic assignment and related problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 16:173-187.

[15] Gambardella, L.M., Taillard, E.D. and Dorigo, M. (1999). “Ant colonies for the quadratic assignment problem,” Journal of the Operational Research Society, 50:167-176.

[16] Gamboa, D., Rego, C., and Glover, F. (2006). “Implementation analysis of efficient heuristic algorithms for the traveling salesman problem,” Computers and Operations Research, 33(4):1161-1179.

[17] Glover, F. and Laguna, M. (1997). “Tabu search,” Kluwer Academic Publishers.

[18] Glover, F. (1998). “A template for scatter search and path relinking,” Artificial Evolution, J.-K. Hao et al., eds., LNCS 1363, Springer-Verlag, 13-54.

[19] Hasegawa, M., T. Ikeguchi, and K. Aihara (2000). “Exponential and chaotic neurodynamical tabu searches for quadratic assignment problems,” Control and Cybernetics, 29(3): 773-788.

[20] James, T., Rego, C., and Glover, F. (2005). “Sequential and parallel path-relinking algorithms for the quadratic assignment problem,” IEEE Intelligent Systems, 20(4), 58-65.

[21] Kadłuczka, P., Wala, K. (1995). “Tabu search and genetic algorithms for the generalized graph partitioning problem,” Control and Cybernetics, 24 (4):459–476.

[22] Koopmans, T. and Beckmann, M. (1957). “Assignment problems and the location of economic activities,” *Econometrica*, 25(1) 53-76.

[23] Li, Y, Pardalos, P.M., and Resende, M.G.C. (1994). “A greedy randomized adaptive search procedure for the quadratic assignment problem”, Quadratic assignment and related problems, P.M. Pardalos and H. Wolkowicz, eds., DIMACS Series on Discrete Mathematics and Theoretical Computer Science, 16, 237-261.

[24] Lin, F.–T., Kao, C.–Y. and Hsu, C.–C. (1993). “Applying the genetic approach to simulated annealing in solving some NP-hard problems,” IEEE Transactions on Systems, Man, and Cybernetics, 23, 1752–1767.

[25] Maniezzo, V. and Colorni, A. (1999). “The ant system applied to the quadratic assignment problem,” IEEE Transactions on Knowledge and Data Engineering, 11(5):769-778.

[26] Misevicius, A. (2003). “Genetic algorithm hybridized with ruin and recreate procedure: application to the quadratic assignment problem, “ Knowledge-Based Systems, 16, 261-268.

[27] Misevicius, A. (2004). “An improved hybrid genetic algorithm: new results for the quadratic assignment problem,” Knowledge-Based Systems, 17, 65-73.

[28] Misevicius, A. (2005). “A tabu search algorithm for the quadratic assignment problem,” Computational Optimization and Applications, 30, 95-111.

[29] Oliveira, C.A., Pardalos, P.M., and Resende, M.G.C. (2004). “GRASP with path-relinking for the quadratic assignment problem”, Efficient and experimental algorithms, C.C. Ribeiro and S.L. Martins (Eds.), Lecture Notes in Computer Science, 3059, 356--368, Springer-Verlag.

[30] Pardalos, P.M., Pitsoulis, L.S. and Resende, M.G.C. (1995). “A parallel GRASP implementation for the quadratic assignment problem,” In A. Ferreira and J. Rolim, eds, Parallel Algorithms for Irregular Structured Problems - Irregular'94, 111-130. Kluwer Academic Publishers.

[31] Pepper, J.W., Golden, B.L., and Wasil, E.A. (2002). “Solving the traveling salesman problem with annealing-based heuristics: a computational study,” IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans, 32(1), 72-77.

[32] Polak, G.G. (2005). “On a special case of the quadratic assignment problem with an application to storage-and-retrieval devices,” Annals of Operations Research, 138(1):223-233.

[33] Rego, C. (1998). “Relaxed tours and path ejections for the traveling salesman problem,” European Journal of Operational Research, 106:522-538.

[34] Simic, P.D. (1991). “Constrained nets for graph matching and other quadratic assignment problems,” Neural Computation, 3:268-281.

[35] Skorin-Kapov J. (1990). “Tabu search applied to the quadratic assignment problem,” ORSA Journal on Computing, 2:33-45.

[36] SPEC. (2000). “SPEC benchmark results”, http://www.spec.org.

[37] Stutzle, T., and Dorigo, M. (1999). “ACO Algorithms for the Quadratic Assignment Problem,” In New Ideas for Optimization, D. Corne, M. Dorigo, and F. Glover (Eds.), McGraw-Hill, 33-50.

[38] Taillard, E. (1991). “Robust taboo search for the quadratic assignment problem,” Parallel Computing, 17, 443-455.

[39] Tate, D.E. and Alice E.S. (1995). “A genetic approach to the quadratic assignment problem.” Computers and Operations Research, 22:73–83.

[40] Tseng, L. and Liang, S. (2005). “A hybrid metaheuristic for the quadratic assignment problem,” Computational Optimization and Applications, 34, 85-113.

[41] Wilhelm, M.R. and Ward, T.L. (1987). “Solving quadratic assignment problems by simulated annealing,” IIE Transactions 19, 107-119 (1987).

[42] Zhang, Q., Sun J., and Tsang, E. (2005). “An evolutionary algorithm with guided mutation for the maximum clique problem,” IEEE Transactions on Evolutionary Computation, 9(2), 192-200.