Multistart Tabu Search and Diversification Strategies for the Quadratic Assignment Problem

Tabitha James, César Rego, and Fred Glover

***Abstract*—The quadratic assignment problem (QAP) is a well-known combinatorial optimization problem with a wide variety of applications, prominently including the facility location problem. The acknowledged difficulty of the QAP has made it the focus of many metaheuristic solution approaches. In this paper, we show the benefit of utilizing strategic diversification within the tabu search (TS) framework for the QAP, by incorporating several diversification and multistart TS variants. Computational results for an extensive and challenging set of QAP benchmark test problems demonstrate the ability of our TS variants to improve on a classic TS approach that is one of the principal and most extensively 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 the classic TS method as a subroutine.**

***Index Terms*—Combinatorial optimization, quadratic assignment problem (QAP), tabu search (TS).**

I. Introduction

The quadratic assignment problem (QAP) is an NP-hard combinatorial optimization problem first introduced by Koopmans and Beckmann [44] 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:

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

where f is the flow matrix, d is the distance matrix, p is a permutation vector of n indices of facilities (or locations) mapping a possible assignment of n facilities to n locations, and P is the set of all n-vector permutations. For each pair of assignments $r = p(i)$ and $s = p(j)$ in p, the flow f_{ij} between the two facilities i and j is multiplied by the distance d_{rs} between the two locations r and s. The sum of these terms over all pairs gives the total cost assignment $z(p)$ for the permutation p. The objective is to find a permutation p^* in P of minimum total cost.

Manuscript received February 19, 2008. Current version published April 17, 2009. This paper was recommended by Associate Editor Y. Narahari.

T. James is with the Department of Business Information Technology, Pamplin College of Business, Virginia Polytechnic Institute and State University, Blacksburg, VA 24060 USA (e-mail: tajames@vt.edu).

C. Rego is with the School of Business, University of Mississippi, Oxford, MS 38677 USA (e-mail: crego@bus.olemiss.edu).

F. Glover is with the University of Colorado at Boulder, Boulder, CO 80309 USA, and also with OptTek Systems, Inc., Boulder, CO 80302 USA (e-mail: fred.glover@colorado.edu).

Digital Object Identifier 10.1109/TSMCA.2009.2014556

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. The blackboard wiring problem in electronics, the arrangement of electronic components in printed circuit boards and in microchips, the balancing of turbine runners, the 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 [15]. It is also possible to formulate several other well-known combinatorial optimization problems as QAPs, including the traveling salesman problem (TSP), the maximum clique problem, the linear ordering 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 [10], [20], [29], [41], [67], [68], [70], [76], and [87].

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 [12], simulated annealing [19], [84], threshold accepting [59], genetic algorithms (GAs) [18], [82], tabu search (TS) [8], [57], [77], [81], the greedy randomized adaptive search procedure (GRASP) [48], [65], evolution strategies [58], ant colony optimization (ACO) [28], [51], [80], scatter search [21], and path relinking [39]. Many variations of these approaches are present in the literature of the QAP, e.g., [3], [22]–[24], [26], [37], [49], [55], [56], [61], and [83]. In Section II, we survey the current most advanced 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 TS, embedded within the proposed metaheuristic framework. Most of the better approaches explicitly incorporate some variation of the TS algorithm developed by Taillard [81]. Taillard's robust TS (RTS) algorithm on its own obtains good solutions for the QAP and is very inexpensive in terms of computational time.

In this paper, we examine various TS strategies using the RTS algorithm as a benchmark to demonstrate the contribution of these strategies. The resulting TS approaches demonstrate that high-quality results can be obtained by simple and fast procedures incorporating traditional TS intensification and diversification without requiring the complicated designs introduced in more elaborate hybrid metaheuristics. Our approaches are both efficient and easy to implement. Consequently, our improved TS algorithms can also be easily embedded within more complex metaheuristic procedures such as those described in Section II. Our algorithms are shown to outperform RTS, which is the most commonly embedded algorithm in hybrid metaheuristics for the QAP. We demonstrate improved results over several other algorithms considered to follow the multistart framework from the literature, including GRASP and iterative local search (ILS). Competitive results are also obtained against some of the best hybrid metaheuristics for the QAP.

The remainder of this paper is organized as follows. Section II gives an overview of solution approaches for the QAP. Section III provides a discussion of TS and the details of the principal components in a classical TS implementation for the QAP. In Section IV, we present a detailed discussion of our diversification and multistart TS variants. A discussion of the computational results is given in Section V, followed by the concluding remarks in Section VI.

II. QAP Solution Approaches

The QAP was first introduced in 1957 [44], and since then, it has been extensively studied. Sahni and Gonzales [75] showed that the QAP is NP-complete and that no polynomial-time heuristic can have a bounded performance ratio unless $P = NP$. Queyranne [69] further extended this finding to include cases where the distance matrix satisfied the triangle inequality condition. The QAP remains one of the hardest combinatorial optimization problems and, due to its computational complexity, has spawned decades of research in both exact and approximation algorithms. Although significant advances have been made in the development of exact solution methods for the QAP, the success of these methods are still limited to solving rather small instances of the QAP using massively parallel computing environments that are cost prohibitive to most. Due to these computational limitations and the inherent complexity of the problem, the application of heuristic approaches to the QAP has been a popular area of research for several decades. Heuristics provide a way to obtain good but not-guaranteed-optimal solutions to hard problems within reasonable computational times. In this section, we briefly review some successful approaches that are most relevant to our study. We begin our discussion in Section II-A by tracing the milestone developments in the area of exact methods for the QAP. In Section II-B, we examine methods that may be expressly classed as multistart algorithms. Then, Section II-C provides coverage of the leading population-based algorithms from the literature.

A. Exact Methods

The most successful exact solution methods for the QAP are branch-and-bound algorithms based on different bounding techniques [7], [14], [30], [47]. A detailed review of the various bounding techniques can be found in surveys by Pardalos *et al.* [66] and Anstreicher [4]. The complexity of the QAP is such that the exact solution of problems of size $20 \leq n \leq 30$ is considered computationally nontrivial. The solution of larger problems, where $30 \leq n \leq 40$, usually requires the use of massively parallel programming environments. Since the introduction of some of the first exact solution methods for the QAP by Lawler [47], several notable advances toward the optimal solution of larger problem instances have occurred.

The exact solution of any QAP instances of size $n = 20$ was not obtained until the mid-1990s. Notably, Mautor and Roucairol [54] presented for the first time the exact optimal solution to the classical nug16 instance, as well as els19 and a size-20 problem by Armour and Buffa. Clausen and Perregaard [16] were the first to solve the classical nug20 instance to optimality using a parallel branch-and-bound algorithm. The solution of nug20 required approximately 960 min of computational time and 16 processors.

Using a dynamic programming algorithm, Marzetta and Brüngger [53] solved the nug25 instance to optimality for the first time. Their algorithm was implemented in parallel and executed on between 64 and 128 processors of an Intel Paragon and nine DEC Alpha processors. The solution was obtained after 30 days of total computation time. A rather more efficient algorithm was then developed by Anstreicher and Brixius [5] using a convex quadratic programming relaxation within a branch-and-bound algorithm. The algorithm was run on a large grid computer and provided the optimal solution for the nug25 instance after 6.7 h of wall-clock time. This would have amounted to approximately 13 days of CPU time on the sequential counterpart.

Hahn and Krarup [36] solved the QAP instance kra30a after approximately 99 days of computational time on a sequential workstation. Nystrom [60] reported the optimal solutions to two problems of size 36 (ste36b and ste36c) using a distributed programming environment with 22 processors. The solution took approximately 60 and 200 days of computation time for ste36b and ste36c, respectively.

Perhaps the most acclaimed development in the exact solution of QAP instances is due to the work of Anstreicher *et al.* [6], as the authors reported the exact solution to nug30. With the same branch-and-bound algorithm used to solve nug25 in [5], enhanced with additional branching rules, they were able to solve the nug27, nug28, nug30, kra30b, and tho30 instances to optimality. The solution of nug30 took approximately seven days to complete on a computational grid with an average of 650 computers simultaneously processing, a time that would translate to almost seven years of computation on a single fast computer workstation. A similar equivalence results in 15 years of computation to solve the tho30 instance.

B. Multistart Methods

As the name implies, a multistart method is one that executes multiple times from different initial settings. In the context of heuristic search, a multistart can be generally described as a method that iterates between two component methods: 1) a constructive method used to create a new starting solution and 2) an improvement method that attempts to improve this solution by ILS. A multistart algorithm iterates between these two component methods (or procedures) while saving the best solution found throughout the search. In its simplest form, a multistart can simply be viewed as a random restart algorithm in which a local search is iteratively run on multiple randomly generated solutions. Multistarts may become more sophisticated with the use of adaptive search strategies to both construct the initial solution and guide the local search component. Unlike random restart algorithms, adaptive multistart methods rely on a more strategic construction of starting solutions for posterior iterative improvement based on the results of previous optimizations. See [11] for a general exposition of the method.

As noted in [73], when a constructive process is built upon information gathered from previous constructing iterations, it can, without lost of generality, be interpreted as a type of improvement method where the search for new solutions is given by a constructive neighborhood. A constructive neighborhood successively adds new components to create a new solution, while keeping some components of the current solution fixed. These include methods that assemble components from different solutions and methods that alter the customary choices of the constructive process. In general, mechanisms used to influence choices in constructive processes proceed by changing algorithm parameters, evaluating solution components, or restricting the selection to a specified set of elements. Once a complete solution is obtained in a multistart algorithm, the method switches to a type of transition neighborhood to implement what is usually called a local search procedure, which iteratively moves from one solution to another based on the definition of a neighborhood structure. As we will see later, population-based neighborhoods, commonly used in evolutionary procedures, may also appear in multistart approaches. Under these adaptive frameworks, a stand-alone constructive process may only be used to initiate the algorithm since all subsequent restarts derive from adaptive modifications of some existing solutions through appropriate destructive/constructive neighborhoods or by local perturbations generated by some transition neighborhood.

The purpose of restarting is to drive the search into new regions of the solution space, which may be viewed as a form of diversification. Although in adaptive forms of multistart methods the constructive and improvement components may be seemingly tied together in a unified search process, it is worth mentioning that even when perturbation mechanisms are used to restart the search from a new diverse solution, its neighborhood structure is usually different from the one considered in the actual improvement component. Perturbation mechanisms of this kind have proved useful in multistart algorithms for combinatorial optimization. A prominent example is the multistart (or "iterative") variant of the well-known Lin–Kernighan procedure for TSP. In this setting, the so-called "double-bridge" neighborhood is used to restart the algorithm once a local optimum had been found—see, e.g., [17], [40], and [52]. (A similar approach has been recently applied to the maximum clique problem in [42].)

The concept of repeatedly perturbing a solution based upon the search history and reapplying the local search component has been popularized in the field of metaheuristics as ILS and clearly defines a type of multistart strategy based on perturbation. See [50] for a comprehensive exposition of the method and a survey of perturbation techniques used in a variety of applications.

In general, the degree of perturbation is influenced by the type of neighborhood and the number of moves applied at each restart. A similar analog can be made to reflect the degree of reconstruction in destructive/constructive neighborhoods. Obviously, more than one alternative neighborhood can be used alone or in combination to induce different levels of diversification in both types of processes. A particularly successful application of combining different neighborhoods has recently proved effective in a local search algorithm for job shop scheduling [72]. In this implementation, the algorithm restarts multiple times from a diversified solution produced by a local optimization procedure that removes a specified number of machines (solution components) from the schedule (solution) and constructs a new solution by iteratively reinserting the machines into the schedule.

A typical characteristic of multistart algorithms is the use of some degree of randomization as a means to induce diversification at each restart. Random-start approaches represent an extreme of high degree of randomization. Such an approach may be characterized as inserting diversity into a search in an uncontrolled manner. On the other hand, the types of adaptive multistart approaches that make use of randomization employ it in a controlled manner.

GRASP is one of the best known examples of an adaptive randomized multistart algorithm. The constructive phase of GRASP is an example of the application of a more systematic application of randomness. A GRASP implementation for the QAP is presented in [48]. The constructive phase of the GRASP algorithm for the QAP creates a sorted list of elements based on the costs of the assignments and then constructs an initial solution by randomly selecting elements from the portion of the list that falls within a defined percentage. Hence, this construction phase is not completely greedy but applies a level of controlled randomness.

As discussed in [33], improved forms of adaptive multistarts can be achieved by drawing on principles of adaptive memory of TS. In this manner, memory may be used to construct a solution based on known components of previously good solutions. In a similar manner, memory may be used to introduce diversification. That is, by using memory, an initial solution may be generated such that it is by some known degree different from the best solution(s). Our diversified best-solution-found TS (DivTS) algorithm in this paper follows this approach by utilizing a diversity procedure to strategically create an initial solution from good solutions found throughout the search. In this way, diversity can be introduced in a controlled manner as opposed to the use of uncontrolled randomization.

Fleurent and Glover [27] have proposed the use of memory in a multistart TS for the QAP. They suggest the application of the type of frequency and recency memory common in adaptive memory approaches to the constructive component of the multistart algorithm. Their multistart introduced intensification into the algorithm by using the concepts of strongly determined and consistent variables in the construction phase. Strongly determined variables are those elements of the solution that cannot be removed without significantly degrading the objective function value. Consistent variables are those elements of a solution that tend to be common across high-quality solutions. In strategies employing these concepts, certain variables are fixed based on search guidance. In addition to the study of

Fleurent and Glover, Palubeckis [62] also developed a multistart TS variant that employed these techniques to unconstrained binary quadratic optimization. Palubeckis [62] explored several other multistart TS variants for unconstrained binary quadratic optimization in his study, including a random restart, a constructive procedure, a randomized semigreedy construction, and a perturbation procedure. Palubeckis [63] also explored the same type of techniques applied to the graph-coloring problem, and Palubeckis and Krivickiene [64] explored the same type of techniques applied to the MAX-CUT problem. All of these variants differ in the approach taken to generate the starting solutions and have shown good results for the problems to which they are applied.

A more advanced variant of GRASP enhanced by a TS path-relinking strategy is introduced in [61] for the QAP. This algorithm is similar to the GRASP algorithm of Li *et al.* [48] mentioned above, except that it modifies the improvement procedure of the algorithm by exploring search paths between solutions using transition neighborhoods in a path-relinking fashion.

Misevicius [57] also introduced several multistart TS variants (ETS1, ETS2, and ETS3) for the QAP. These algorithms simply apply a modified RTS procedure of Taillard's algorithms [81] 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. Several diversifying perturbation schemes were incorporated into these algorithms, 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). The variants test various combinations of these operators. Misevicius's approach can be viewed as a multistart TS using a variety of diversification operators. The layering to these TSs are more complicated than those in this paper, as often, several levels of restarting occur with multiple diversification operators.

Stützle [79] examined several ILS algorithms for the QAP. All of his variants used a simple two-opt local search and perturb the solution using random pairwise exchanges. To determine a solution from which to restart the search, several options were considered. In the traditional ILS variant, the best solution, which may or may not be the working solution obtained by the current run of the local search, is perturbed, and then, a local search is applied (ILS1). In the second version, a random restart (ILS2) was employed, which straightforwardly replaces the working solution with a random permutation. The third variant always perturbs the working solution obtained from the local search (ILS3). The fourth variant allows worse solutions based upon a probability, which are then perturbed, and the local search is restarted (ILS4).

ACO may also be considered a multistart algorithm. Like GRASP, ACO uses a probabilistic construction phase but differs by using the search history to influence it. The premise of ACO algorithms is that as a search occurs and good solutions are found, more of the ants will take similar paths (that is, locate elements in the same position). The collection of this search information is stored in memory and called the pheromone trail. ACO constructively builds solutions by choosing an assignment influenced by the pheromone trail. A local search is then applied to the constructed solution. Several ACO algorithms for the QAP are given by Stützle and Dorigo [80]. The first variant modifies the construction phase to use the pheromone trail to modify the current solution rather than construct a new one (ACO1). The next two variants use a typical ACO construction phase, but ACO2 applies a two-opt local search, whereas ACO3 uses RTS as its local search.

Another type of multistart appears in some variants of the relaxation adaptive memory programming (RAMP) approach [71]. Here, restarts occur in the dual space of a relaxation of the original problem whose solutions provide starting points for the construction of primal feasible solutions (i.e., solutions that are feasible to the original problem). Once a solution is constructed, a local search is then applied to find enhanced solutions. The method alternates between the primal (improvement) and the dual (constructive) procedures while maintaining appropriate memory structures that affect both the primal and the dual searches. Simple variants of RAMP keep track of the best solution found during the search; however, more advanced variants maintain a reference set of high-quality and diverse solutions, which may be combined in an evolutionary fashion to generate new enhanced solutions. This more advanced setting is an example of using population-based neighborhoods within a multistart approach. Although relatively new, the RAMP method has already proved very effective in solving a number of combinatorial optimization problems, which include set covering, facility location, linear ordering, generalized assignment, and network design.

This is by no means an exhaustive list of possible adaptive memory strategies that may be employed in a multistart algorithm. For further uses of adaptive memory strategies and their use in restarted algorithms, see, for example, [1], [9], [34], and [45].

C. Leading Population-Based Methods for the QAP

A number of the recent metaheuristic solution techniques consist of GA variants coupled with TS. Misevicius proposed two GA approaches joined with TS for the QAP, a GA hybridized with a "ruin-and-recreate" procedure (GA/TS) [55] and an improved hybrid GA (GA/TS/I) utilizing a "shift mutation operator" [56]. Both of these approaches are superimposed on a modified version of Taillard's TS, 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 TS 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 [55]. The more advanced hybrid GA version of this approach adds a "shift mutation" operator that further perturbs selected solutions to create greater diversification [56].

Two other hybrid GA approaches are given by Drezner [23], [24], which similarly incorporate several TS variations within a modified GA framework. Drezner [23] examines the use of a descent heuristic (GA/SD), a simple TS (GA-S/TS), and a new "concentric" TS procedure (GA-C/TS), which identifies and evaluates candidate moves based upon their distance from a "center" solution. Drezner [24] incorporates an extension of the concentric TS approach (GA-IC/TS) that considers a larger number of permissible moves.

A population-based path-relinking approach using Taillard's RTS procedure as an improvement method is introduced and studied in [39]. Recently, a hybrid metaheuristic approach combining ACO with a GA and local search (ACO-GA/LS) has been proposed by Tseng and Liang [83]. Stützle [79] presents several population-based variants of ILS (ILS5, ILS6, and I-ILS6). These algorithms maintain a population of solutions and use ILS to operate on the population. The third variant, I-ILS6, uses an improved local search from all the previous ILS algorithms discussed.

Due to differences in the QAP test sets chosen for testing, direct comparisons of the techniques outlined above are difficult. This makes the designation of a "best" metaheuristic impossible. The best performing algorithms from the literature typically provide computational results for different test instances from QAPLIB/Taillard's repository. The hybrid GAs introduced by Misevicius [55], [56] and the TS variants [57] produce some of the best solutions for the Taillard test sets, but they are not run on any other test instances. The hybrid GA of Drezner [24] produces the best quality solutions for the Skorin–Kapov test suite. No results are presented for the Taillard test instances for Drezner's algorithms.

Although 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. As RTS is often a component of more complex metaheuristics, the development of a multistart TS variant that would provide better solution quality than RTS in similar runtimes is the goal of this paper.

III. TS FOR THE QAP

The hallmark of TS is the use of adaptive memory to guide the exploration of the search space. Basic (rudimentary) TS procedures make use of short-term 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 TS 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.

It is well known that the design of appropriate intensification and diversification strategies is essential to achieve high levels of performance in local search algorithms. There are many different ways to design and implement these strategies in TS, and their use typically depends on the application. In general, the first level of intensification and diversification is achieved by changing the tabu list size. Small sizes encourage the exploration of solutions near a local optimum, and larger sizes push the search out of the vicinity of the local optimum. Varying the tabu list size during the search provides one way to explore the effect of tabu list size, and this approach has proved useful in a number of TS applications. Advanced forms of short-term memory may consider various types of tabu restrictions associated with several aspiration criteria, which may be used to make a decision about the acceptance of the tabu status of a particular move. Taillard's RTS algorithm [81] considered in this paper is a classical example of an effective TS short-term memory algorithm using multiple levels of aspiration criteria. Although in some cases, the use of short-term memory may be sufficient to produce high-quality solutions, many studies have shown that versions of TS that include longer term memory components generally prove superior to more limited versions (e.g., [25], [38], [43], [46], and [78]). The literature in TS abounds with algorithms exploring intensification and diversification through a variety of long-term memory structures and strategies applied to numerous problem settings. (A simple Google Scholar search returns over 1000 articles on TS containing the exact phrase "long term memory.") These strategies can range from relatively simple to very intricate designs, depending on the application and the type of implementation chosen to explore possible tradeoffs between ease of implementation and sophistication of the search.

The most commonly used long-term memory keeps track of the frequency of components of solutions (or move attributes) that occur in high-quality (or elite) solutions or that have been involved in selected moves. As part of a longer term intensification strategy, the elements of a solution may be judiciously selected to be provisionally locked into the solution on the basis of having occurred with high frequency in the best solutions found. Conversely, diversification strategies typically penalize such highest frequency elements so that the search can move into previously unexplored regions of the solution space (see [34] for a detailed explanation of various forms and uses of memory within TS processes).

Although intensification and diversification are equally important components in finding the best solutions, the design of an effective diversification strategy is usually a significantly more complex task. Whereas intensification restricts the search to regions containing attributes that are known to be part of high-quality solutions, diversification is concerned with discovering new regions of potentially good solutions in a vastly larger search space of unknown characteristics.

Diversification techniques generally fall into two main categories. The first, called continuous diversification, alters the regular search trajectory by perturbing the current search parameters or by biasing the evaluation of possible moves. The second, called discontinuous diversification, strategically replaces the current working solution with a different solution, using it as a new starting point to continue the search. TS strategies that combine solution attributes (e.g., by hash functions [86], chunking [85], or vocabulary building methods [35]) or complete solutions (e.g., by path relinking [31]) to create other solutions that are used as starting points to generate other solutions may be viewed as discontinuous diversification approaches. Moreover, when the construction of the new solutions produced by these or other TS strategies alternate with the improvement of all or some of the newly created solutions, such approaches fall into a special type of adaptive multistart methods generally called multistart TS methods, as defined in Section II-B. The path-relinking implementation of James *et al.* [39] is an example of such multistart TS strategy.

For the purpose of this paper, we focus primarily on the simple TS strategies, utilizing intensification and diversification processes that are straightforward and easy to implement.

The RTS algorithm developed by Taillard [81] provides a core TS 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 diversification and multistart TS procedures we have developed. The remainder of this section describes the basic components of our approaches.

A. Neighborhood Specification

The neighborhood most commonly employed in local search algorithms for the QAP, including RTS and the multistart methods presented in this paper, is the classical two-exchange (or swap) neighborhood.

To illustrate, consider the following permutation:

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

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

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

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 $(k > 2)$ has limited their use. Exceptions are the very-large-scale-neighborhood search procedures [2] and advanced neighborhood constructions derived from ejection chain methods [74].

In the case of the simple two-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 I. 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 two-exchange moves on a given permutation, thereby saving computational cost, as we examine next.

B. Computation and Update of Solution Cost Changes

To expedite the evaluation of a two-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. Let the triplet (p, r, s) define a move that swaps the elements r and s in a permutation p; the partial costs associated with the move, for symmetrical QAP instances, can be calculated by

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

Let us consider an example for the symmetric case. Suppose d and f are given as

$$d = \begin{bmatrix} 10 & 3 & 8 & 4 \\ 3 & 10 & 6 & 1 \\ 8 & 6 & 10 & 9 \\ 4 & 1 & 9 & 10 \end{bmatrix} \quad f = \begin{bmatrix} 2 & 4 & 1 & 7 \\ 4 & 2 & 3 & 5 \\ 1 & 3 & 2 & 7 \\ 7 & 5 & 7 & 2 \end{bmatrix}.$$

For illustration purposes, we choose an arbitrary permutation $p = (2, 1, 3, 4)$ and calculate the objective function $z(p) = 344$ using (1).

Using permutation p, we can calculate the partial cost matrix Δ. For each off-diagonal value in Δ, $\Delta(p, r, s)$ denotes the change in cost of performing the exchange (r, s) on p. Therefore, in our example, the off-diagonal elements in Δ are calculated using (2) as follows:

$$\Delta(p,1,2) = 2 \times ((8-6) \times (1-3) + (4-1) \times (7-5)) = 4$$
$$\Delta(p,1,3) = 2 \times ((3-6) \times (1-4) + (4-9) \times (7-5)) = -2$$
$$\Delta(p,1,4) = 2 \times ((3-1) \times (7-4) + (8-9) \times (7-3)) = 4$$
$$\Delta(p,2,3) = 2 \times ((3-8) \times (3-4) + (1-9) \times (7-7)) = 10$$
$$\Delta(p,2,4) = 2 \times ((3-4) \times (5-4) + (6-9) \times (7-1)) = -38$$
$$\Delta(p,3,4) = 2 \times ((8-4) \times (5-3) + (6-1) \times (7-1)) = 76$$

which results in the following partial cost matrix:

$$\Delta = \begin{bmatrix} & 4 & -2 & 4 \\ 4 & & 10 & -38 \\ -2 & 10 & & 76 \\ 4 & -38 & 76 & \end{bmatrix}.$$

Considering the partial cost matrix Δ, the best exchange on p is (2, 4), resulting in the following permutation $t = (2, 4, 3, 1)$ and objective function $z(t) = 344 - 38 = 306$.

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

$$\Delta(t, u, v) = \Delta(p, u, v) + 2(f_{ru} - f_{rv} + f_{sv} - f_{su}) \times \left(d_{t(s)t(u)} - d_{t(s)t(v)} + d_{t(r)t(v)} - d_{t(r)t(u)}\right) \quad (3)$$

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 (2) can be used to compute the cost (see [81] for a more detailed discussion of these costs and [13] for a discussion of asymmetrical cases).

Continuing the previous example, to update the partial cost matrix Δ using (2) when u or v is the same as r or s and (3) otherwise, assuming that we made the exchange (2, 4)

$$\Delta(t,1,2)=2\times((8-6)\times(7-3)+(4-1)\times(7-4))=34$$

$$\Delta(t,1,3)=\ -2+(2\times((3-6+9-4)\times(4-1+7-5)))=18$$

$$\Delta(t,1,4)=2\times((3-1)\times(7-5)+(8-9)\times(1-3))=12$$

$$\Delta(t,2,3)=2\times((3-8)\times(3-5)+(1-9)\times(1-7))=116$$

$$\Delta(t,2,4)=2\times((3-4)\times(4-5)+(6-9)\times(1-7))=38$$

$$\Delta(t,3,4)=2\times((8-4)\times(4-3)+(6-1)\times(7-7))=8.$$

As can be seen above, (3) is only used for $\Delta(t,1,3)$ since in our small example, this is the only case where the condition is not violated. In other words, in all the other cases, either u or v is equal to either 2 or 4. As the problem size grows, it can be seen that (3) will be more frequently used. The new partial cost matrix becomes

$$\Delta=\begin{bmatrix} & 34 & 18 & 12\\ 34 & & 116 & 38\\ 18 & 116 & & 8\\ 12 & 38 & 8 & \end{bmatrix}.$$

For every exchange performed, the partial cost matrix Δ is updated as demonstrated using permutation t. The process restarts from the beginning for an entirely new permutation.

C. 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 this paper, we adopt the rules for designating an exchange tabu utilized in the RTS algorithm. However, our variants change 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 our algorithms, which result in a tabu list with a different structure than that in RTS.

To determine the 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 from 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.

D. 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 paper are the same as those used in the RTS algorithm.

```
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
```

Fig. 1. TS framework.

The aspiration criteria utilized in this paper 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 second establishes whether 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.

E. Traditional TS Algorithm

The outline of a traditional TS algorithm is provided in Fig. 1. The outer loop of Fig. 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 the solution quality, execution time, or iterations. The stopping condition utilized in all variations of the TS algorithms in this paper, including the RTS used for comparisons, is to stop the execution of the algorithm after the global best solution 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. In this paper, it was modified in the original RTS to allow for valid comparisons with the proposed multistart variants.

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 Fig. 1, the resulting algorithm is an RTS with a modified stopping condition. The diversification and multistart variants developed in this paper change the structure of this skeleton and will be discussed in depth in the following sections.

IV. Diversification and Multistart TS Strategies

The algorithm skeleton provided in Fig. 1 assumes that the algorithm is seeded with one initial permutation and continuously modifies the current permutation until a stopping criterion is met. A multistart approach extends this traditional design by adjusting the search strategy to modify the standard search path. A continuous diversification can be obtained by strategically changing the current parameter settings while continuing the search from the current working solution. Conversely, a discontinuous diversification strategy consists of restarting the search from a solution that somewhat differs from the current working solution. This may consist of going back to a previously visited solution to explore a different search trajectory or creating a new solution to restart the search.

As discussed earlier, the type of procedures that discontinue the current search trajectory by employing some degree of reconstruction of the solution structure or explicit perturbation are generally called multistart approaches. In this paper, we explore discontinuous diversification strategies that both change the trajectory from a previously visited solution and create a new working solution, as well as continuous diversification strategies.

Accordingly, the TS variants developed in this paper can be separated into two categories. The first category implements a continuous diversification TS process and contains variants that perturb the search by modifying some algorithm parameters but continue the exploration from the same working solution. By modifying the parameters, 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 these 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 TS (RDTS) and the tabu tenure modification TS (TTMTS) both fall into this category, where the alteration of the tabu list matrix and the tabu tenure parameters is explored. The tabu list matrix contains the current tabu status of the elements that establish whether 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 algorithms.

```
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 diversification
            Reset allowable_failures
        End If
    End If
End Loop
```

Fig. 2. Diversification and multistart TS framework.

The second category implements a discontinuous diversification TS process and contains the random-restart TS (RRTS), the best-solution-found TS (BSFTS), and 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 a randomly drawn solution, the global best solution, or a diversified version of the global best solution, respectively. Although 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 previously visited solution or a variation of such a solution that is already known to be favorable, thus utilizing information from the previous iterations of the search.

The addition of a diversification component changes the structure of the algorithm presented in Fig. 1 by forcefully altering the search trajectory in some manner when an undesirable amount of stagnation occurs. Each variant follows the skeleton given in Fig. 2 and differs only in the type of diversification applied. The diversification process that occurs when the allowable failure threshold is reached for each of the five variants will be detailed in the following sections.

A. *RDTS Approach*

This TS variant implements a continuous diversification approach simply by modifying the tabu tenure parameter. As the search begins to stagnate (the allowable failure condition in Fig. 2 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 failure condition is met again or the stopping condition is reached. In this manner, a restricted descent from the current working solution occurs at the diversification stage 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 diversification applied in this variant is only in terms of a parameter adjustment; the current working solution is not replaced. The intention of this variant is to gauge the impact of releasing the tabu list to benchmark the more sophisticated modifications.

B. TTMTS Approach

The TTMTS variant takes the previous continuous diversification 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 value and a lower value are 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 failure condition is reached, not only is the tabu list released, but also new upper and lower values to determine the tabu tenure range are chosen. At each diversification stage, 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 tenure 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 diversification stage. 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 diversification stage.

C. RRTS Approach

The RRTS variant implements a discontinuous diversification strategy by multistart. The algorithm 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, but runs several times for a shorter number of iterations.

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.

D. BSFTS Restart Approach

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 discontinuous diversification 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.

```
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
```

Fig. 3. Pseudocode for the diversification method.

E. DivTS Restart Approach

The DivTS variant also releases the tabu list and modifies the tabu tenure parameters in the same manner as above. The DivTS discontinuous diversification variant again differs in the replacement of the current working solution at the restart. This variant perturbs the search trajectory by replacing 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 greatly differ 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 best known solution (BKS) is described in the following section.

Diversification Procedure: The diversification procedure used in DivTS is suggested by Glover [31]. The pseudocode for this procedure is given in Fig. 3. The method generates new starting solutions from a randomly generated seed solution in the manner illustrated below.

Suppose that a randomly generated permutation such as s is given, where $s(1), \ldots, s(n)$ are the locations to which facilities $1, \ldots, n$ 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 Fig. 3). For example, if step $= 3$, then through the first pass of the inner loop, start $= 3$, which results in the partial solution

$$ss = (5, 3, 12, 4).$$

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

$$ss = (5, 3, 12, 4, 1, 9, 2, 6).$$

TABLE I
TS VARIANT PARAMETER SETTINGS

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

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

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

This method can be used to generate 1 to n solutions from any given permutation, where n is the number of locations/facilities 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.

V. Computational Results and Discussion

For computational testing, a set of 126 test problems obtained from QAPLIB (http://www.seas.upenn.edu/qaplib/inst.html) and Taillard's repository (http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html) were used. All of the TS variants proposed in this paper and the RTS algorithm 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 ten times, starting from ten different randomly generated seed solutions (initial working solutions).

A. Computational Analysis of the Variants

To determine the best TS algorithms, we tested our five variants against RTS on a subset of the test instances. The most widely used 31 instances were chosen as a representative set of diverse and difficult instances in the standard testbed. The Skorin–Kapov instances and the Taillard instances are the most common instances utilized for computational testing in recent studies. Furthermore, these two test sets contain a greater number of large instances, and obtaining the BKS often proves more elusive for these two test sets. The best variant found in this analysis was then run on the remaining 95 instances, and comparisons to algorithms from the literature are given in Section V-B.

The parameters for each variation are shown in Table I. The stopping condition was set to $50\,000^{*}n$ (n being the problem size or number of facilities/locations for the problem). The RTS procedure was written identically to Taillard's algorithm with the exception of the 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 diversification and multistart variants.

The allowable failure limits define a range from which the number of iterations, in which no improving move is found before the diversification is applied, is chosen. An initial number of iterations are chosen from this range at the beginning of the algorithm, and in each diversification stage, a new value is chosen. The upper limit of this range is set to be less than the maximum failure 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 TS variants, with the exception of the RDTS variant, modified these parameter values during the diversification phase. The new tabu tenure range parameters, during the diversification stage only, were drawn from the range of values given in Table I. 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.

Table II provides a comparison of all the variants, as well as the RTS algorithm. The average percent deviation (APD) from the BKS is given for each algorithm, as well as the number of times the BKS was found out of the ten runs (in parentheses) and the average time to completion. All times are in minutes. The best overall average deviation for each test problem is bolded. An asterisk denotes that the variant did as well as or better than the RTS algorithm. The averages over all problems for each variant and RTS are also provided in the last row.

As can be seen in Table II, all five variants outperformed RTS. The worst variant, RDTS, still obtained solution quality as good as or better than RTS on 25 of the 31 test instances. TTMTS and RRTS tied with or outperformed RTS on 29 of the 31 problems. The best two variants, BSFTS and DivTS, did better than or as well as RTS on 30 of the 31 test instances. The averages over all test instances are: 0.165 for RTS, 0.154 for RDTS, 0.120 for TTMTS, 0.124 for RRTS, 0.119 for BSFTS, and 0.112 for DivTS. All five proposed TS variants obtain better average solution quality than RTS. The running times of all algorithms are very similar across all test instances. On the average, these times range from 39.31 (for RTS) to 46.93

TABLE II
COMPARATIVE ANALYSIS BETWEEN TS VARIANTS AND RTS

		RTS		RDTS		TTMTS		RRTS		BSFTS		DivTS	
Problem	BKS	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time
sko42	15812	$\mathbf{0.000}_{(10)}$	3.88	$\mathbf{0.000}^{*}_{(10)}$	4.05	$\mathbf{0.000}^{*}$	3.86	$\mathbf{0.000}^{*}_{(10)}$	3.91	$\mathbf{0.000}^{*}_{(10)}$	3.74	$\mathbf{0.000}^{*}_{(10)}$	3.98
sko49	23386	$0.038_{(2)}$	8.38	$0.027^{*}_{(3)}$	11.30	0.004^{*}	9.60	$\mathbf{0.000}^{*}_{(10)}$	10.20	$0.014^{*}_{(7)}$	8.52	$0.008^{*}_{(7)}$	9.61
sko56	34458	$0.010_{(4)}$	16.75	$0.015^{*}_{(3)}$	13.72	$\mathbf{0.002}^{*}$	13.75	$\mathbf{0.002}^{*}_{(8)}$	13.85	$\mathbf{0.002}^{*}_{(8)}$	14.22	$\mathbf{0.002}^{*}_{(8)}$	13.16
sko64	48498	$0.005_{(5)}$	23.97	$0.008_{(2)}$	25.21	$\mathbf{0.000}^{*}$	25.44	$\mathbf{0.000}^{*}_{(10)}$	21.43	$\mathbf{0.000}^{*}_{(10)}$	24.00	$\mathbf{0.000}^{*}_{(10)}$	22.03
sko72	66256	0.043	37.38	0.043	30.42	0.014^{*}	35.29	0.016^{*}	41.41	$0.013^{*}_{(1)}$	34.50	$\mathbf{0.006}^{*}_{(2)}$	37.98
sko81	90998	0.051	59.25	0.044^{*}	45.65	0.017^{*}	62.36	$0.019^{*}_{(1)}$	55.33	$0.023^{*}_{(1)}$	56.25	$\mathbf{0.016}^{*}_{(2)}$	56.36
sko90	115534	0.062	65.60	0.044^{*}	74.12	0.017^{*}	99.69	$0.019^{*}_{(1)}$	93.38	$\mathbf{0.013}^{*}_{(1)}$	92.08	0.026^{*}	89.60
sko100a	152002	0.089	76.40	0.081^{*}	100.32	0.026^{*}	134.53	$0.036^{*}_{(1)}$	114.87	$\mathbf{0.024}^{*}_{(2)}$	132.80	0.027^{*}	129.22
sko100b	153890	0.056	108.80	0.054^{*}	81.93	0.011^{*}	124.84	$\mathbf{0.006}^{*}_{(4)}$	124.15	$0.010^{*}_{(2)}$	113.02	$0.008^{*}_{(2)}$	106.55
sko100c	147862	0.031	93.84	0.025^{*}	105.05	0.008^{*}	113.95	$0.007^{*}_{(2)}$	115.43	0.010^{*}	107.90	$\mathbf{0.006}^{*}_{(2)}$	126.69
sko100d	149576	0.055	111.97	$0.038^{*}_{(1)}$	100.30	0.016^{*}	129.23	0.014^{*}	122.70	$\mathbf{0.011}^{*}_{(3)}$	121.98	$0.027^{*}_{(1)}$	123.45
sko100e	149150	0.041	95.80	0.046	100.25	$\mathbf{0.007}^{*}$	130.14	$\mathbf{0.007}^{*}_{(1)}$	117.43	0.011^{*}	115.62	$0.009^{*}_{(1)}$	108.84
sko100f	149036	0.066	100.28	0.057^{*}	110.01	0.021^{*}	118.90	0.034^{*}	118.57	$\mathbf{0.012}^{*}_{(1)}$	127.00	0.023^{*}	110.28
tai20a	703482	$\mathbf{0.000}_{(10)}$	0.24	$\mathbf{0.000}^{*}_{(10)}$	0.23	$\mathbf{0.000}^{*}_{(10)}$	0.23	$\mathbf{0.000}^{*}_{(10)}$	0.23	$\mathbf{0.000}^{*}_{(10)}$	0.24	$\mathbf{0.000}^{*}_{(10)}$	0.24
tai25a	1167256	$\mathbf{0.000}_{(10)}$	0.55	$\mathbf{0.000}^{*}_{(10)}$	0.51	$\mathbf{0.000}^{*}_{(10)}$	0.50	$\mathbf{0.000}^{*}_{(10)}$	0.61	$\mathbf{0.000}^{*}_{(10)}$	0.51	$\mathbf{0.000}^{*}_{(10)}$	0.56
tai30a	1818146	$\mathbf{0.000}_{(10)}$	1.65	$\mathbf{0.000}^{*}_{(10)}$	1.58	$\mathbf{0.000}^{*}_{(10)}$	1.41	$\mathbf{0.000}^{*}_{(10)}$	1.48	$\mathbf{0.000}^{*}_{(10)}$	1.51	$\mathbf{0.000}^{*}_{(10)}$	1.31
tai35a	2422002	$0.112_{(5)}$	2.92	$0.069^{*}_{(6)}$	3.76	$0.056^{*}_{(6)}$	3.16	$0.027^{*}_{(8)}$	3.48	$0.073^{*}_{(6)}$	3.60	$\mathbf{0.000}^{*}_{(10)}$	4.44
tai40a	3139370	0.462	4.25	0.412^{*}	5.12	0.284^{*}	5.22	0.269^{*}	4.61	0.311^{*}	4.81	$\mathbf{0.222}^{*}_{(1)}$	5.16
tai50a	4938796	0.882	10.08	0.801^{*}	12.98	$\mathbf{0.700}^{*}$	10.07	0.774^{*}	11.45	0.685^{*}	10.46	0.725^{*}	10.23
tai60a	7205962	0.974	24.73	0.950^{*}	24.49	0.820^{*}	25.92	0.872^{*}	22.00	0.752^{*}	20.65	$\mathbf{0.718}^{*}$	25.69
tai80a	13515450	1.065	54.74	1.004^{*}	68.32	0.817^{*}	69.21	0.822^{*}	62.15	0.841^{*}	54.61	$\mathbf{0.753}^{*}$	52.74
tai100a	21059006	1.071	114.55	1.017^{*}	107.56	0.846^{*}	145.26	0.870^{*}	128.61	0.848^{*}	129.73	$\mathbf{0.825}^{*}$	142.06
tai20b	122455319	$\mathbf{0.000}_{(10)}$	0.23	$\mathbf{0.000}^{*}_{(10)}$	0.23	$\mathbf{0.000}^{*}_{(10)}$	0.23	$\mathbf{0.000}^{*}_{(10)}$	0.23	$\mathbf{0.000}^{*}_{(10)}$	0.23	$\mathbf{0.000}^{*}_{(10)}$	0.23
tai25b	344355646	$\mathbf{0.000}_{(10)}$	0.46	$\mathbf{0.000}^{*}_{(10)}$	0.46	$\mathbf{0.000}^{*}_{(10)}$	0.46	$\mathbf{0.000}^{*}_{(10)}$	0.46	$\mathbf{0.000}^{*}_{(10)}$	0.46	$\mathbf{0.000}^{*}_{(10)}$	0.46
tai30b	637117113	$\mathbf{0.000}_{(10)}$	1.38	$\mathbf{0.000}^{*}_{(10)}$	1.28	$\mathbf{0.000}^{*}_{(10)}$	1.28	$\mathbf{0.000}^{*}_{(10)}$	1.27	$\mathbf{0.000}^{*}_{(10)}$	1.30	$\mathbf{0.000}^{*}_{(10)}$	1.31
tai35b	283315445	$\mathbf{0.000}_{(10)}$	2.20	$\mathbf{0.000}^{*}_{(10)}$	2.36	$\mathbf{0.000}^{*}_{(10)}$	2.44	$\mathbf{0.000}^{*}_{(10)}$	2.31	$\mathbf{0.000}^{*}_{(10)}$	2.44	$\mathbf{0.000}^{*}_{(10)}$	2.39
tai40b	637250948	$\mathbf{0.000}_{(10)}$	3.20	$\mathbf{0.000}^{*}_{(10)}$	3.17	$\mathbf{0.000}^{*}_{(10)}$	3.17	$\mathbf{0.000}^{*}_{(10)}$	3.18	$\mathbf{0.000}^{*}_{(10)}$	3.18	$\mathbf{0.000}^{*}_{(10)}$	3.18
tai50b	458821517	$\mathbf{0.000}_{(8)}$	11.02	$\mathbf{0.000}^{*}_{(10)}$	9.98	$\mathbf{0.000}^{*}_{(8)}$	9.54	$\mathbf{0.000}^{*}_{(10)}$	8.68	$\mathbf{0.000}^{*}_{(10)}$	8.63	$\mathbf{0.000}^{*}_{(10)}$	8.82
tai60b	608215054	$\mathbf{0.000}_{(6)}$	17.89	$\mathbf{0.000}^{*}_{(4)}$	18.91	$\mathbf{0.000}^{*}_{(4)}$	18.56	$\mathbf{0.000}^{*}_{(7)}$	24.41	$\mathbf{0.000}^{*}_{(8)}$	19.48	$\mathbf{0.000}^{*}_{(8)}$	17.08
tai80b	818415043	0.008	51.99	0.009	52.36	$0.013_{(2)}$	49.02	$0.012_{(2)}$	58.68	$\mathbf{0.006}^{*}$	57.29	$\mathbf{0.006}^{*}$	58.24
tai100b	1185996137	$\mathbf{0.008}_{(3)}$	114.16	0.015	134.23	$0.040_{(2)}$	107.63	0.036	145.96	$0.044_{(1)}$	123.15	0.056	118.91
Average		$0.165_{(123)}$	39.31	$0.154_{(119)}$	40.32	$0.120_{(150)}$	46.93	$0.124_{(155)}$	46.21	$0.119_{(151)}$	44.96	$0.112_{(154)}$	44.86

(for TTMTS). The average for the most effective algorithm across the board (DivTS) is 44.86 s.

Out of the continuous diversification category of TS variants, TTMTS is the best, outperforming RDTS on 18 of the test instances. Of the remaining 13 instances, TTMTS and RDTS tied on 11, and RDTS won only on two. TTMTS (0.120) had a better overall average than RDTS (0.154). Furthermore, TTMTS found 150 BKSs whereas RDTS found only 119. TTMTS obtained the best overall solution to 15 instances, and RDTS obtained the best overall solution to 11 instances. It is interesting to note that TTMTS performed quite well for a simple TS, obtaining the unique overall solution to one instance (Tai50a). It even outperformed the RRTS variant from the second category in overall average solution quality: 0.120 (TTMTS) to 0.124 (RRTS).

In the discontinuous diversification category, DivTS was the best performing algorithm. DivTS outperformed RRTS on 12 of the 31 test instances and tied on 13. RRTS outperformed DivTS on the remaining six instances. DivTS outperformed BSFTS on 12 instances and tied on 14. BSFTS obtained better solution quality on five instances. DivTS outperformed both RRTS and BSFTS in overall average solution quality: DivTS (0.112), BSFTS (0.119), and RRTS (0.124). RRTS obtained one more BKS than DivTS: 155 to 154. BSFTS obtained 151 BKSs, slightly less than the other two variants. DivTS obtained the best overall solution to 22 instances versus RRTS's 16 instances and BSFTS's 18 instances. DivTS obtained the most unique overall best APDs, i.e., on eight instances (Tai35a, Tai40a, Tai60a, Tai80a, Tai100a, Sko72, Sko81, and Sko100c). BSFTS obtained unique overall best solutions for four instances: Sko90, Sko100a, Sko100d, and Sko100f. RRTS obtained unique overall best solutions for three instances: Sko49, Sko100b, and Sko100e.

All but one of the unique overall best solutions was found by the variants in the second category. Overall, the variants in the second category performed better than those in the first category, with the previously noted exception that TTMTS obtained a better average than RRTS over all instances. All of our TS variants are shown to outperform RTS. The times for all algorithms are similar. Of interest is that our TS variants do not significantly increase the runtimes yet improve the solution quality.

The main purpose of this paper was to identify a TS variant that would provide better performance than the widely used RTS algorithm. All five variants proposed in this paper are shown to outperform RTS. Second, we want to identify the most successful TS variant among those proposed. DivTS is clearly the overall winner, as it significantly outperforms RTS and the other variants. It outperforms its closest two competitors by more than double the number of instances. The average over all 31 instances is lower than all other variants, and it finds double the number of best overall unique APDs of the other variants.

B. Extended Computational Analysis

We took the best variant, DivTS, and extended the test set to include 95 additional instances obtained from both QAPLIB and the problem instances maintained by Taillard. This test set comprises almost all classical QAP instances and allows for additional comparisons of our best multistart TS to the algorithms from the literature. In particular, we are able to compare our performance to GRASP, ACO, and ILS approaches, which are similar in nature to the multistart TS algorithm. Comparisons to several of the best more complex metaheuristics from the literature are also given. To clearly demonstrate the performance of our algorithm on the different types of instances, we adopt the

TABLE III
LONGER RUN RESULTS FOR DivTS (PROBLEM TYPES 1 AND 2)

Problem	BKS	APD	BPD	Time	Problem	BKS	APD	BPD	Time
Type 1 – Unstructured, Randomly Generated					**Type 2 - Instances with Grid-Distances**				
tai20a	703482	$0.000_{(10)}$	0.000	0.24	nug12	578	$0.000_{(10)}$	0.000	0.40
tai25a	1167256	$0.000_{(10)}$	0.000	0.56	nug14	1014	$0.000_{(10)}$	0.000	0.70
tai30a	1818146	$0.000_{(10)}$	0.000	1.31	nug15	1150	$0.000_{(10)}$	0.000	0.90
tai35a	2422002	$0.000_{(10)}$	0.000	4.44	nug16a	1610	$0.000_{(10)}$	0.000	0.11
tai40a	3139370	$0.222_{(1)}$	0.000	5.16	nug16b	1240	$0.000_{(10)}$	0.000	0.11
tai50a	4938796	0.725	0.607	10.23	nug17	1732	$0.000_{(10)}$	0.000	0.14
tai60a	7205962	0.718	0.551	25.69	nug18	1930	$0.000_{(10)}$	0.000	0.16
tai80a	13515450	0.753	0.434	52.74	nug20	2570	$0.000_{(10)}$	0.000	0.23
tai100a	21059006	0.825	0.703	142.06	nug21	2438	$0.000_{(10)}$	0.000	0.26
rou12	235528	$0.000_{(10)}$	0.000	0.40	nug22	3596	$0.000_{(10)}$	0.000	0.31
rou15	354210	$0.000_{(10)}$	0.000	0.90	nug24	3488	$0.000_{(10)}$	0.000	0.40
rou20	725522	$0.000_{(10)}$	0.000	0.24	nug25	3744	$0.000_{(10)}$	0.000	0.45
lipa20a	3683	$0.000_{(10)}$	0.000	0.23	nug27	5234	$0.000_{(10)}$	0.000	0.58
lipa20b	27076	$0.000_{(10)}$	0.000	0.23	nug28	5166	$0.000_{(10)}$	0.000	1.45
lipa30a	13178	$0.000_{(10)}$	0.000	1.20	nug30	6124	$0.000_{(10)}$	0.000	1.21
lipa30b	151426	$0.000_{(10)}$	0.000	1.19	sko42	15812	$0.000_{(10)}$	0.000	3.98
lipa40a	31538	$0.000_{(10)}$	0.000	3.16	sko49	23386	$0.008_{(7)}$	0.000	9.61
lipa40b	476581	$0.000_{(10)}$	0.000	3.15	sko56	34458	$0.002_{(8)}$	0.000	13.16
lipa50a	62093	$0.000_{(10)}$	0.000	6.57	sko64	48498	$0.000_{(10)}$	0.000	22.03
lipa50b	1210244	$0.000_{(10)}$	0.000	6.48	sko72	66256	$0.006_{(2)}$	0.000	37.98
lipa60a	107218	$0.000_{(10)}$	0.000	14.56	sko81	90998	$0.016_{(2)}$	0.000	56.36
lipa60b	2520135	$0.000_{(10)}$	0.000	14.17	sko90	115534	0.026	0.007	89.60
lipa70a	169755	$0.000_{(10)}$	0.000	23.29	sko100a	152002	0.027	0.009	129.22
lipa70b	4603200	$0.000_{(10)}$	0.000	22.42	sko100b	153890	$0.008_{(2)}$	0.000	106.55
lipa80a	253195	$0.000_{(10)}$	0.000	54.61	sko100c	147862	$0.006_{(2)}$	0.000	126.69
lipa80b	7763962	$0.000_{(10)}$	0.000	34.34	sko100d	149576	$0.027_{(1)}$	0.000	123.45
lipa90a	360630	$0.137_{(7)}$	0.000	83.57	sko100e	149150	$0.009_{(1)}$	0.000	108.84
lipa90b	12490441	$0.000_{(10)}$	0.000	51.49	sko100f	149036	0.023	0.005	110.28
					scr12	31410	$0.000_{(10)}$	0.000	0.40
					scr15	51140	$0.000_{(10)}$	0.000	0.90
					scr20	110030	$0.000_{(10)}$	0.000	0.23
					tho30	149936	$0.000_{(10)}$	0.000	1.20
					tho40	240516	$0.001_{(9)}$	0.000	4.60
					tho150	8133398	0.030	0.006	487.19
					wil50	48816	$0.000_{(10)}$	0.000	7.92
					wil100	273038	$0.005_{(1)}$	0.000	104.56
Average		0.121	0.082	20.17			0.005	0.001	43.12

TABLE IV
LONG-RUN DivTS RESULTS CONTINUED (PROBLEM TYPES 3 AND 4)

Problem	BKS	APD	BPD	Time	Problem	BKS	APD	BPD	Time	Problem	BKS	APD	BPD	Time
Type 3 - Real-Life Instances					**Real-Life Instances (continued)**					**Type 4 - Real-Life Like Instances**				
bur26a	5426670	$0.000_{(10)}$	0.000	0.52	kra30a	88900	$0.000_{(10)}$	0.000	1.21	tai20b	122455319	$0.000_{(10)}$	0.000	0.23
bur26b	3817852	$0.000_{(10)}$	0.000	0.52	kra30b	91420	$0.000_{(10)}$	0.000	1.21	tai25b	344355646	$0.000_{(10)}$	0.000	0.46
bur26c	5426795	$0.000_{(10)}$	0.000	0.51	kra32	88700	$0.000_{(10)}$	0.000	1.38	tai30b	637117113	$0.000_{(10)}$	0.000	1.31
bur26d	3821225	$0.000_{(10)}$	0.000	0.52	ste36a	9526	$0.000_{(10)}$	0.000	2.30	tai35b	283315445	$0.000_{(10)}$	0.000	2.39
bur26e	5386879	$0.000_{(10)}$	0.000	0.51	ste36b	15852	$0.000_{(10)}$	0.000	2.22	tai40b	637250948	$0.000_{(10)}$	0.000	3.18
bur26f	3782044	$0.000_{(10)}$	0.000	0.52	ste36c	8239110	$0.000_{(10)}$	0.000	2.27	tai50b	458821517	$0.000_{(10)}$	0.000	8.82
bur26g	10117172	$0.000_{(10)}$	0.000	0.52	esc16a	68	$0.000_{(10)}$	0.000	0.11	tai60b	608215054	$0.000_{(8)}$	0.000	17.08
bur26h	7098658	$0.000_{(10)}$	0.000	0.52	esc16b	292	$0.000_{(10)}$	0.000	0.11	tai80b	818415043	0.006	0.001	58.24
els19	17212548	$0.000_{(10)}$	0.000	0.19	esc16c	160	$0.000_{(10)}$	0.000	0.11	tai100b	1185996137	0.056	0.005	118.91
had12	1652	$0.000_{(10)}$	0.000	0.40	esc16d	16	$0.000_{(10)}$	0.000	0.11					
had14	2724	$0.000_{(10)}$	0.000	0.70	esc16e	28	$0.000_{(10)}$	0.000	0.11					
had16	3720	$0.000_{(10)}$	0.000	0.11	esc16f	0	$0.000_{(10)}$	0.000	0.11					
had18	5358	$0.000_{(10)}$	0.000	0.16	esc16g	26	$0.000_{(10)}$	0.000	0.11					
had20	6922	$0.000_{(10)}$	0.000	0.23	esc16h	996	$0.000_{(10)}$	0.000	0.11					
chr12a	9552	$0.000_{(10)}$	0.000	0.40	esc16i	14	$0.000_{(10)}$	0.000	0.11					
chr12b	9742	$0.000_{(10)}$	0.000	0.40	esc16j	8	$0.000_{(10)}$	0.000	0.11					
chr12c	11156	$0.000_{(10)}$	0.000	0.40	esc32a	130	$0.000_{(10)}$	0.000	1.40					
chr15a	9896	$0.000_{(10)}$	0.000	0.90	esc32b	168	$0.000_{(10)}$	0.000	1.38					
chr15b	7990	$0.000_{(10)}$	0.000	0.90	esc32c	642	$0.000_{(10)}$	0.000	1.37					
chr15c	9504	$0.000_{(10)}$	0.000	0.82	esc32d	200	$0.000_{(10)}$	0.000	1.37					
chr18a	11098	$0.000_{(10)}$	0.000	0.17	esc32e	2	$0.000_{(10)}$	0.000	1.37					
chr18b	1534	$0.000_{(10)}$	0.000	0.16	esc32g	6	$0.000_{(10)}$	0.000	1.37					
chr20a	2192	$0.000_{(10)}$	0.000	0.32	esc32h	438	$0.000_{(10)}$	0.000	1.37					
chr20b	2298	$0.200_{(9)}$	0.000	0.38	esc64a	116	$0.000_{(10)}$	0.000	17.36					
chr20c	14142	$0.000_{(10)}$	0.000	0.23	esc128	64	$0.000_{(10)}$	0.000	163.02					
chr22a	6156	$0.000_{(10)}$	0.000	0.36										
chr22b	6194	$0.152_{(8)}$	0.000	0.56										
chr25a	3796	$0.943_{(5)}$	0.000	0.98										
Average							0.024	0.000	4.05			0.007	0.001	23.40

problem classification given by Stützle [79]. This classification groups the QAP instances into four categories.

1) Unstructured, randomly generated: This group contains instances in which the distance matrix was randomly generated based on a uniform distribution.
2) Grid-based distance matrix: This group contains instances in which the distances are the Manhattan distance between points on a grid.
3) Real-life instances: These instances are obtained from real-life QAP applications.
4) Real-life-like instances: These instances were generated to appear similar to real-life problems.

Tables III and IV present the results of the DivTS algorithm on all 126 instances by category. The results in Tables III and IV are divided by the classification given above into four groups. For each group, the table contains the instance name, the BKS

TABLE V

LONGER RUN DivTS COMPARISONS WITH THE LITERATURE FOR TYPE-1 PROBLEMS

Problem	DivTS		RTS	GRASP	ACO-GA/LS	ETS1	ETS2	ETS3	GA/TS	GA/TS/I	ILS1	ILS2	ILS3	ILS4	ILS5	ILS6	I-ILS6	ACO1	ACO2	ACO3
	APD	BPD	APD	BPD	BPD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD
tai20a	$0.000_{(10)}$	0.000	0.000		0.110	0.000	0.000	0.000	0.061	0.000	0.723	0.503	0.542	0.467	0.500	0.344		0.675	0.191	0.428
tai25a	$0.000_{(10)}$	0.000	0.000		0.290	0.037	0.000	0.015	0.088	0.000	1.181	0.876	0.896	0.823	0.869	0.656	0.000	1.189	0.488	1.751
tai30a	$0.000_{(10)}$	0.000	0.000		0.340	0.003	0.041	0.000	0.019	0.000	1.304	0.808	0.989	1.141	0.707	0.668	0.000	1.311	0.359	1.286
tai35a	$0.000_{(10)}$	0.000	0.112		0.490	0.000	0.000	0.000	0.126	0.000	1.731	1.110	1.113	1.371	1.010	0.901	0.000	1.762	0.773	1.586
tai40a	$0.222_{(1)}$	0.000	0.462		0.590	0.167	0.130	0.173	0.338	0.209	2.036	1.319	1.490	1.491	1.305	1.082	0.280	1.989	0.933	1.131
tai50a	0.725	0.607	0.882		0.850	0.322	0.354	0.388	0.567	0.424	2.127	1.496	1.491	1.968	1.574	1.211	0.610	2.800	1.236	1.900
tai60a	0.718	0.551	0.974		0.030	0.570	0.603	0.677	0.590	0.547	2.200	1.498	1.692	2.081	1.622	1.349	0.820	3.070	1.372	2.484
tai80a	0.753	0.434	1.065		0.860	0.321	0.390	0.405	0.271	0.320	1.775	1.198	1.200	1.576	1.219	1.029	0.620	2.689	1.134	2.103
tai100a	0.825	0.703	1.071		0.800	0.367	0.371	0.441	0.296	0.259							0.690			
rou12	$0.000_{(10)}$	0.000	0.000	0.000	0.000															
rou15	$0.000_{(10)}$	0.000	0.000	0.000	0.000															
rou20	$0.000_{(10)}$	0.000	0.000	0.000	0.000															
lipa20a	$0.000_{(10)}$	0.000	0.000	0.000	0.000															
lipa20b	$0.000_{(10)}$	0.000	0.000	0.000	0.000															
lipa30a	$0.000_{(10)}$	0.000	0.000	0.000	0.000															
lipa30b	$0.000_{(10)}$	0.000	0.000	0.000	0.000															
lipa40a	$0.000_{(10)}$	0.000	0.000	0.000	0.000												0.000			
lipa40b	$0.000_{(10)}$	0.000	0.000	0.000	0.000															
lipa50a	$0.000_{(10)}$	0.000	0.000	0.905	0.000												0.000			
lipa50b	$0.000_{(10)}$	0.000	0.000	0.000	0.000															
lipa60a	$0.000_{(10)}$	0.000	0.000	0.839	0.000												0.000			
lipa60b	$0.000_{(10)}$	0.000	0.000	0.000	0.000												0.000			
lipa70a	$0.000_{(10)}$	0.000	0.000	0.746	0.060												0.000			
lipa70b	$0.000_{(10)}$	0.000	0.000	0.000	0.000												0.000			
lipa80a	$0.000_{(10)}$	0.000	0.363	0.676	0.550															
lipa80b	$0.000_{(10)}$	0.000	0.000	0.000	0.000												0.000			
lipa90a	$0.137_{(7)}$	0.000	0.341	0.615	0.350															
lipa90b	$0.000_{(10)}$	0.000	0.000	0.000	0.000												0.000			
Average	0.121	0.082	0.188	0.199	0.190	0.199	0.210	0.233	0.262	0.195	1.635	1.101	1.177	1.365	1.101	0.905	0.189	1.936	0.811	1.584
DivTS Average			0.121	0.000	0.082	0.360	0.360	0.360	0.360	0.360	0.302	0.302	0.302	0.302	0.302	0.302	0.203	0.302	0.302	0.302

for the instance, the APD over ten runs of the algorithm (the number of times the BKS was found is given in parentheses), the best percent deviation (BPD) out of the ten runs, and the average solution times in minutes. Categories 1 and 2 are given in Table III, and categories 3 and 4 are given in Table IV. The averages over each category of instances are provided at the bottom of each table.

The results presented in Tables III and IV show that DivTS obtains high-quality results for all test instances in relatively short runtimes. The algorithm obtains APDs of 0.05 or below for all but ten of the 126 test problems. The BKS is found 1073 out of 1260 times (approximately 85% of the total runs). All but six of the average runtimes are less than 2 h, with anything under 90 facilities/locations running in less than 1 h.

DivTS performed quite well on all four categories of test instances. The worst overall APD (0.121) was for the type-1 test instances. This was due to the Tai*a instances, as the algorithm always found the BKS on all runs for all but six of the 28 test instances. The six instances with greater than 0.000 APDs were the larger Tai*a instances and Lipa90a. For the Lipa90a instance, the BKS was still obtained in seven out of the ten runs. There are 47 type-1 instances, with the average number of facilities/locations (or n) being approximately 47.

For the type-2 instances, DivTS performed very well. The overall APD for this category was 0.005, and the overall average BPD was 0.001. The algorithm did not always find the BKS for some of the Sko* instances, the two Tho instances, and the size-100 Wil instance. This category of instances contained some of the biggest instances, with the average size of the instances being approximately 50 facilities/locations. There were 36 instances in this category.

The type-3 category contained the largest number of test instances. There are 53 instances in this category, with the average size being approximately 25 facilities/locations. Although this category contains the greatest number of instances, it also has the smallest average problem size. DivTS performed well on this category, with an overall APD of 0.024. The average BPD for this test set was 0.000, which indicates that the BKS was always found at least once for every test instance in the category. In fact, the BKS was always found at least half the time on all instances.

The last category contained the type-4 instances and was the smallest category. This category contained only the Tai*b instances. The algorithm performed well on this category as well, with an overall APD of 0.007 and an average BPD of 0.001. The average size (number of facilities/locations) of this category was approximately 49, and there were nine instances.

Tables V–VIII give comparisons of the DivTS algorithm to the following additional methods from the literature:

- GRASP [48];
- ACO-GA/LS [83];
- GA hybrid with a strict descent operator (GA/SD) [23];
- GA hybrid with a simple TS operator (GA-S/TS) [23];
- GA hybrid with concentric TS Operator (GA-C/TS) [23];
- GA hybrid with an improved concentric TS operator (GA/IC-TS) [24];
- three TS variants (ETS1, ETS2, and ETS3) [57];
- two GA hybrids with TS (GA/TS and GA/TS/I) [55], [56];
- four ILS variants (ILS1, ILS2, ILS3, and ILS4) [79];
- two population-based ILS algorithms (ILS5 and ILS6) [79];
- improved population-based ILS algorithm (I-ILS6) [79];
- three ACO variants (ACO1, ACO2, and ACO3) [80].

Tables V–VIII provide comparisons of the DivTS variant with some of the best approaches from the literature, as well as several other algorithms that may be classified as multistarts. The DivTS algorithm is highly competitive with these algorithms in terms of both 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. Although 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

TABLE VI
LONGER RUN DivTS COMPARISONS WITH THE LITERATURE FOR TYPE-2 PROBLEMS

Problem	DivTS		RTS	GRASP	ACO-GA/LS	GA/SD	GA-S/TS	GA-C/TS	GA-IC/TS	ILS1	ILS2	ILS3	ILS4	ILS5	ILS6	I-ILS6	ACO1	ACO2	ACO3
	APD	BPD	APD	BPD	BPD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD
nug12	$0.000_{(10)}$	0.000	0.000	0.000	0.000														
nug14	$0.000_{(10)}$	0.000	0.000		0.000														
nug15	$0.000_{(10)}$	0.000	0.000	0.000	0.000														
nug16a	$0.000_{(10)}$	0.000	0.000		0.000														
nug16b	$0.000_{(10)}$	0.000	0.000		0.000														
nug17	$0.000_{(10)}$	0.000	0.000		0.000														
nug18	$0.000_{(10)}$	0.000	0.000		0.000														
nug20	$0.000_{(10)}$	0.000	0.000	0.000	0.000														
nug21	$0.000_{(10)}$	0.000	0.000		0.000														
nug22	$0.000_{(10)}$	0.000	0.000		0.000														
nug24	$0.000_{(10)}$	0.000	0.000		0.000														
nug25	$0.000_{(10)}$	0.000	0.000		0.000											0.000			
nug27	$0.000_{(10)}$	0.000	0.000		0.000											0.000			
nug28	$0.000_{(10)}$	0.000	0.000		0.000											0.000			
nug30	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.013	0.001	0.000		0.219	0.020	0.052	0.020	0.013	0.007	0.000	0.098	0.026	0.042
sko42	$0.000_{(10)}$	0.000	0.000		0.000	0.014	0.001	0.000		0.269	0.010	0.010	0.161	0.022	0.000	0.000	0.076	0.015	0.104
sko49	$0.008_{(7)}$	0.000	0.038		0.060	0.107	0.062	0.009		0.226	0.133	0.133	0.139	0.090	0.068	0.000	0.141	0.067	0.150
sko56	$0.002_{(8)}$	0.000	0.010		0.010	0.054	0.007	0.001	0.000	0.418	0.087	0.087	0.153	0.102	0.071	0.000	0.101	0.068	0.118
sko64	$0.000_{(10)}$	0.000	0.005		0.000	0.051	0.019	0.000	0.000	0.413	0.068	0.068	0.202	0.079	0.057	0.000	0.129	0.042	0.171
sko72	$0.006_{(2)}$	0.000	0.043		0.020	0.112	0.056	0.014	0.000	0.383	0.134	0.134	0.294	0.139	0.085	0.000	0.277	0.109	0.243
sko81	$0.016_{(2)}$	0.000	0.051		0.030	0.087	0.058	0.014	0.003	0.586	0.101	0.100	0.194	0.100	0.082	0.001	0.144	0.071	0.223
sko90	0.026	0.007	0.062		0.040	0.139	0.073	0.011	0.001	0.576	0.131	0.187	0.322	0.262	0.128	0.007	0.231	0.192	0.288
sko100a	0.027	0.009	0.089		0.020	0.114	0.070	0.018	0.002	0.358	0.115	0.161	0.257	0.191	0.109	0.006			
sko100b	$0.008_{(2)}$	0.000	0.056		0.010	0.096	0.042	0.011	0.000							0.012			
sko100c	$0.006_{(2)}$	0.000	0.031		0.000	0.075	0.045	0.003	0.001							0.007			
sko100d	$0.027_{(1)}$	0.000	0.055		0.030	0.137	0.084	0.049	0.000							0.002			
sko100e	$0.009_{(1)}$	0.000	0.041		0.000	0.071	0.028	0.002	0.000							0.021			
sko100f	0.023	0.005	0.066		0.030	0.148	0.110	0.032	0.003							0.037			
scr12	$0.000_{(10)}$	0.000	0.000	0.000	0.000														
scr15	$0.000_{(10)}$	0.000	0.000	0.000	0.000														
scr20	$0.000_{(10)}$	0.000	0.000	0.000	0.000														
tho30	$0.000_{(10)}$	0.000	0.000		0.000	0.009	0.000	0.000								0.000			
tho40	$0.001_{(9)}$	0.000	0.011		0.010	0.069	0.042	0.010	0.003							0.000			
tho150	0.030	0.006	0.078		0.090											0.068			
wil50	$0.000_{(10)}$	0.000	0.010		0.000	0.038	0.011	0.002	0.000							0.000			
wil100	$0.005_{(1)}$	0.000	0.028		0.010	0.076	0.043	0.002	0.000							0.004			
Average	0.005	0.001	0.019	0.000	0.010	0.078	0.042	0.010	0.001	0.383	0.089	0.104	0.194	0.111	0.067	0.008	0.150	0.074	0.167
DivTS Average			0.005	0.000	0.001	0.009	0.009	0.009	0.011	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.007	0.007	0.007

TABLE VII
LONGER RUN DivTS COMPARISONS WITH THE LITERATURE FOR TYPE-3 PROBLEMS

Problem	DivTS		RTS	GRASP	ACO-GA/LS	GA/SD	GA-S/TS	GA-C/TS	GA-IC/TS	GA/TS	GA/TS/I	ILS1	ILS2	ILS3	ILS4	ILS5	ILS6	I-ILS6	ACO1	ACO2	ACO3
	APD	BPD	APD	BPD	BPD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD
bur26a-h	0.000	0.000	0.000	0.000	0.000							0.000	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.006	0.000
els19	$0.000_{(10)}$	0.000	0.000	0.000	0.000					0.000	0.000										
had*	0.000	0.000	0.000		0.000																
chr12a	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
chr12b	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
chr12c	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
chr15a	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
chr15b	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
chr15c	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
chr18a	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
chr18b	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
chr20a	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
chr20b	$0.200_{(9)}$	0.000	0.400	3.133	0.870																
chr20c	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
chr22a	$0.000_{(10)}$	0.000	0.000	0.552	0.000																
chr22b	$0.152_{(8)}$	0.000	0.213	1.421	0.000																
chr25a	$0.943_{(5)}$	0.000	0.601	4.636	0.000					0.232	0.000							0.000			
kra30a	$0.000_{(10)}$	0.000	0.000	0.000	0.000					0.224	0.000							0.000			
kra30b	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.253	0.089	0.000		0.028	0.000							0.000			
kra32	$0.000_{(10)}$	0.000	0.000			0.037	0.019	0.000										0.000			
ste36a	$0.000_{(10)}$	0.000	0.000	0.651	0.000				0.000	0.061	0.000							0.000			
ste36b	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.246	0.149	0.005		0.000	0.000							0.000			
ste36c	$0.000_{(10)}$	0.000	0.000	0.188	0.000	0.015	0.000	0.000	0.000	0.001	0.000							0.000			
esc16a	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.142	0.066	0.039													
esc16b	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
esc16c	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
esc16d	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
esc16e	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
esc16f	$0.000_{(10)}$	0.000	0.000	0.000	0.000							3.862	2.745	9.521	1.860	1.264	1.475		0.630	0.134	0.314
esc16g	$0.000_{(10)}$	0.000	0.000	0.000	0.000							0.672	0.090	0.000	0.134	0.000	0.000		0.071	0.023	0.049
esc16h	$0.000_{(10)}$	0.000	0.000	0.000	0.000							0.094	0.026	0.046	0.051	0.031	0.008				
esc16i	$0.000_{(10)}$	0.000	0.000	0.000	0.000																
esc16j	$0.000_{(10)}$	0.000	0.000	0.000	0.000							0.377	0.099	0.451	0.227	0.071	0.015			0.036	0.181
esc32a	$0.000_{(10)}$	0.000	0.000	0.000	0.000					0.154	0.000	0.000	0.000	0.000	0.000	0.000	0.000			0.000	0.000
esc32b	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000										0.000			
esc32c	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000										0.000			
esc32d	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000										0.000			
esc32e	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000										0.000			
esc32g	$0.000_{(10)}$	0.000	0.000	0.000	0.000													0.000			
esc32h	$0.000_{(10)}$	0.000	0.000	0.000	0.000													0.000			
esc64a	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000		0.000	0.000							0.000			
esc128	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000		0.000	0.000							0.000			
Average	0.024	0.000	0.029	0.265	0.021	0.063	0.029	0.004	0.000	0.070	0.000	0.834	0.494	1.670	0.379	0.228	0.250	0.000	0.234	0.040	0.109
DivTS Average			0.024	0.000	0.000	0.000	0.000	0.000	0.000	0.105	0.105	0.000	0.000	0.000	0.000	0.000	0.000	0.063	0.000	0.000	0.000

TABLE VIII
LONGER RUN DivTS COMPARISONS WITH THE LITERATURE FOR TYPE-4 PROBLEMS

Problems	DivTS		RTS	ACO-GA/LS	ETS1	ETS2	ETS3	GA/TS	GA/TS/I	ILS1	ILS2	ILS3	ILS4	ILS5	ILS6	I-ILS6	ACO1	ACO2	ACO3
	APD	BPD	APD	BPD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD
tai20b	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.045	0.000	0.045	0.000	0.000	0.000	0.000	0.091	0.000	0.000
tai25b	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.000	0.000	0.007	0.000	0.000	0.000	0.007	0.000	0.000	0.000	0.000	0.000	0.000	0.000
tai30b	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.093	0.000	0.000	0.000	0.000	0.000	0.000	0.000
tai35b	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.019	0.000	0.059	0.000	0.131	0.049	0.081	0.000	0.000	0.000	0.000	0.026	0.051	0.000
tai40b	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.204	0.000	0.000	0.000	0.000	0.000	0.402	0.000
tai50b	$0.000_{(10)}$	0.000	0.000	0.000	0.000	0.003	0.000	0.002	0.000	0.203	0.185	0.282	0.028	0.042	0.033	0.000	0.192	0.172	0.002
tai60b	$0.000_{(8)}$	0.000	0.000	0.000	0.000	0.001	0.003	0.000	0.000	0.029	0.059	0.645	0.023	0.005	0.000	0.000	0.048	0.005	0.005
tai80b	0.006	0.001	0.008	0.000	0.008	0.036	0.016	0.003	0.000	0.785	0.256	0.703	0.260	0.222	0.383	0.000	0.667	0.591	0.096
tai100b	0.056	0.005	0.008	0.010	0.072	0.123	0.034	0.014	0.000	0.219	0.096	0.711	0.202	0.113	0.083	0.000			
Average	0.007	0.001	0.002	0.001	0.009	0.020	0.006	0.009	0.000	0.157	0.072	0.308	0.057	0.042	0.055	0.000	0.128	0.153	0.013
DivTS Average			0.007	0.001	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.001	0.001	0.001

unable to run all algorithms on the same test problems, using the same hardware.

The reported runtimes over the set of algorithms used for comparisons dramatically vary. The differing hardware, compilers, and programming languages used make true comparisons of runtimes impossible. Therefore, runtimes are not reported in the tables. However, the parameters of the DivTS algorithm were purposefully set to be within the range of computational times reported by the algorithms used for comparison. This compromise means that our algorithm will not produce surprising runtimes compared with previous algorithms from the literature. It can be seen from the previous tables, where our results are presented, that our runtimes are not significantly different from those of the RTS algorithm, which, as mentioned, is a popular technique in the literature. This indicates that the execution time for our algorithm is reasonable.

Each of Tables V–VIII contains one category of test instances following the classification given above. The name of the test instance is given in the first column of each table. The APD and BPD obtained by DivTS for each instance directly follows. The APD for each algorithm used for comparison is provided in the remaining columns of the tables. In some cases, BPD was reported rather than APD in this paper. In those instances, the column heading indicates this occurrence. Since we have both APD and BPD for our test runs, we can make comparisons using the correct data. Not all algorithms utilized the same test set, so comparisons are only shown for the overlapping instances. Blank spaces indicate that results were not provided for that instance by the corresponding algorithm. The average solution quality over all the instances run by the corresponding algorithm is provided at the bottom of each table.

Considering the algorithms that may be classified as multistart algorithms, DivTS did very well in comparison. We have already demonstrated the improved performance of DivTS over the RTS algorithm. The ETS algorithms only report results for nine of the type-1 problems and all of the type-4 instances (a total of 18 of the 126 instances). DivTS meets or exceeds the solution quality of the ETS algorithm in four of the nine type-1 instances that can be compared. The ETS algorithms obtain better solution quality on the remaining five type-1 instances. ETS3 slightly edges out DivTS on one of the nine type-4 instances. However, DivTS's average solution quality over all type-3 instances is better than that of two of the ETS variants. ETS3 slightly edges out DivTS by 0.001 on these instances.

The traditional ILS algorithms, ILS1–ILS4, report results for 32 of the 126 instances. For the type-1 and type-2 instances where comparisons can be made, DivTS obtains much better solution quality on all instances. On type-3 and type-4 instances, DivTS meets or exceeds the solution quality of the traditional ILS algorithms on all comparable instances. The ACO algorithms report results on 29 of the 126 problems. The comparable instances include several from each classification. Again, DivTS meets or exceeds the solution quality on all comparable instances against all three ACO variants.

GRASP reported results for 77 of the test instances from type-1, type-2, and type-3 problems. The GRASP study gave only BPDs. The quality of the best solution obtained by DivTS meets or exceeds the GRASP algorithm for all 77 comparable instances. In fact, in several cases, our average solution quality is better than their BPD. As can be seen, our algorithm is highly competitive with the algorithms that may be classed as multistarts, with only the ETS algorithms obtaining comparable solution quality.

Considering the more complex population-based approaches, DivTS is still competitive. The hybrid ACO/GA/LS algorithm ran 125 of the 126 instances. As with GRASP, we compare the BPD rather than the APD. DivTS meets or exceeds the BPD on all but two of the 125 comparable test instances.

The hybrid GA approaches of Drezner (GA/SD, GA-S/TS, and GA-C/TS) provided results for only 29 out of the 126 instances. These instances are from the type-1 and type-2 categories only. DivTS outperforms Drezner's GA hybrids with the simple TS operator (GA/S-TS) and the strict descent operator (GA/SD) in all cases. The original GA hybridized with the concentric TS (GA/C-TS) wins in eight of the 29 problems. The hybrid method that enhances a GA with an improved concentric TS (GA/IC-TS) outperforms the DivTS algorithm in 11 of the 16 comparable cases. Misevicius's hybrid GAs report results for 28 of the 126 test instances from the type-1, type-3, and type-4 classifications. The original hybrid GA (GA/TS) performs better on seven of the 28 instances. The improved version (GA/TS-I) performs better than DivTS on eight of the 28 instances. Our simple algorithm performs as well as or better than these population-based metaheuristics on at least half of the comparable test instances with the exception of the best hybrid of Drezner (GA/IC-TS).

As previously noted, the platforms and runtimes differ between the algorithms, so the comparisons made here are only to show that our algorithm is highly competitive with some of the best variants from the literature. Without a full result set for all algorithms, it is difficult to determine which, if any, algorithm is superior to all the others over a full set of QAP instances. The results presented for DivTS illustrate that it performs well on all problem sets and is competitive with the algorithms that have been shown to work well. Among the multistart algorithms, DivTS is highly competitive. DivTS even performs well against many of the more complex population-based hybrid algorithms.

VI. Conclusion

We have introduced several new diversification and multistart TS variants for the QAP. The resulting algorithms are shown to improve upon the RTS algorithm commonly used as a subprocedure in metaheuristic approaches for the QAP. Our outcomes 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. We have also uncovered benefits from applying simple intensification and strategic diversification to the search.

The success of the DivTS variant shows the merit of applying strategic diversification rather than relying on randomization. Likewise, the success of the simple adaptive memory techniques used in the diversification and multistart variants discloses the power of using search information and strategic operators to guide the exploration.

Our findings particularly underscore the fact that high-quality results can be obtained from approaches that capitalize on the strategic use of information learned during the search process. The diversification and multistart algorithms examined in this paper are simple modifications of a traditional TS. As such, they quickly execute and are programmatically straightforward. The performance of our algorithms is shown to be highly competitive with that of more complicated hybrid metaheuristic approaches. Our outcomes suggest the promise of further advances by applying greater use of intelligently designed strategies. Additionally, the execution speed and programmatic simplicity of our algorithms make them ideal candidates to use in conjunction with more sophisticated methods such as path relinking or scatter search.

References

[1] S. Ahmadi and I. H. Osman, "Greedy random adaptive memory programming for the capacitated clustering problem," *Eur. J. Oper. Res.*, vol. 162, no. 1, pp. 30–44, Apr. 2005.

[2] R. K. Ahuja, K. C. Jha, J. B. Orlin, and D. Sharma, "Very large-scale neighborhood search for the quadratic assignment problem," *INFORMS J. Comput.*, vol. 19, no. 4, pp. 646–657, 2007.

[3] R. K. Ahuja, J. B. Orlin, and A. Tiwari, "A descent genetic algorithm for the quadratic assignment problem," *Comput. Oper. Res.*, vol. 27, no. 10, pp. 917–934, Sep. 2000.

[4] K. Anstreicher, "Recent advances in the solution of quadratic assignment problems," *Math. Program.*, vol. 97, no. 1/2, pp. 27–42, Jul. 2003.

[5] K. Anstreicher and N. W. Brixius, "Solving quadratic assignment problems using convex quadratic programming relaxations," *Optim. Methods Softw.*, vol. 16, no. 1–4, pp. 49–68, 2001.

[6] K. Anstreicher, N. W. Brixius, J.-P. Goux, and J. Linderoth, "Solving large quadratic assignment problems on computational grids," *Math. Program.*, vol. 91, no. 3, pp. 563–588, Feb. 2002.

[7] A. A. Assad and W. Xu, "On lower bounds for a class of quadratic 0, 1 programs," *Oper. Res. Lett.*, vol. 4, no. 4, pp. 175–180, Dec. 1985.

[8] R. Battiti and G. Tecchiolli, "The reactive tabu search," *ORSA J. Comput.*, vol. 6, no. 2, pp. 126–140, 1994.

[9] R. P. Beausoleil, G. Baldoquin, and R. A. Montejo, "Multi-start and path relinking methods to deal with multiobjective knapsack problems," *Ann. Oper. Res.*, vol. 157, no. 1, pp. 105–133, Jan. 2008.

[10] R. S. Bhaba, E. W. Wilbert, and L. H. Gary, "Locating sets of identical machines in a linear layout," *Ann. Oper. Res.*, vol. 77, pp. 183–207, Jan. 1998.

[11] K. D. Boese, A. B. Kahng, and S. Muddu, "New adaptive multistart techniques for combinatorial global optimizations," *Oper. Res. Lett.*, vol. 16, no. 2, pp. 101–113, Sep. 1994.

[12] C. Bousono-Calzon and M. R. W. Manning, "The hopfield neural network applied to the quadratic assignment problem," *Neural Comput. Appl.*, vol. 3, no. 2, pp. 64–72, Jun. 1995.

[13] R. E. Burkard and F. Rendl, "A thermodynamically motivated simulation procedure for combinatorial optimization problems," *Eur. J. Oper. Res.*, vol. 17, no. 2, pp. 169–174, Aug. 1984.

[14] P. Carraresi and F. Malucelli, "A new lower bound for the quadratic assignment problem," *Oper. Res.*, vol. 40, no. 1, pp. 22–27, Jan./Feb. 1992.

[15] E. Cela, *The Quadratic Assignment Problem: Theory and Algorithms*. Boston, MA: Kluwer, 1998.

[16] J. Clausen and M. Perregaard, "Solving large quadratic assignment problems in parallel," *Comput. Optim. Appl.*, vol. 8, no. 2, pp. 111–127, Sep. 1997.

[17] B. Codenotti, G. Manzini, L. Margara, and G. Resta, "Perturbation: An efficient technique for the solution of very large instances of the Euclidean TSP," *INFORMS J. Comput.*, vol. 8, no. 2, pp. 125–133, 1996.

[18] J. P. Cohoon and W. D. Paris, "Genetic placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-6, no. 6, pp. 956–964, Nov. 1987.

[19] D. T. Connolly, "An improved annealing scheme for the QAP," *Eur. J. Oper. Res.*, vol. 46, no. 1, pp. 93–100, May 1990.

[20] S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil, "See the forest before the trees: Fine-tuned learning and its application to the traveling salesman problem," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 28, no. 4, pp. 454–464, Jul. 1998.

[21] V.-D. Cung, T. Mautor, P. Michelon, and A. Tavares, "Scatter search for the quadratic assignment problem," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1996, pp. 165–169.

[22] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[23] Z. Drezner, "A new genetic algorithm for the quadratic assignment problem," *INFORMS J. Comput.*, vol. 15, no. 3, pp. 320–330, Jul. 2003.

[24] Z. Drezner, "The extended concentric tabu for the quadratic assignment problem," *Eur. J. Oper. Res.*, vol. 160, no. 2, pp. 416–422, Jan. 2005.

[25] J. A. Ferland, S. Ichoua, A. Lavoie, and E. Gagné, "Scheduling using tabu search methods with intensification and diversification," *Comput. Oper. Res.*, vol. 28, no. 11, pp. 1075–1092, Sep. 2001.

[26] C. Fleurent and J. A. Ferland, "Genetic hybrids for the quadratic assignment problem," in *Quadratic Assignment and Related Problems*, ser. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 16, P. Pardalos and H. Wolkowicz, Eds. Baltimore, MD: Amer. Math. Soc., 1994, pp. 173–187.

[27] C. Fleurent and F. Glover, "Improved constructive multi-start strategies for the quadratic assignment problem using adaptive memory," *INFORMS J. Comput.*, vol. 11, no. 2, pp. 198–204, Feb. 1999.

[28] L. M. Gambardella, E. D. Taillard, and M. Dorigo, "Ant colonies for the quadratic assignment problem," *J. Oper. Res. Soc.*, vol. 50, no. 2, pp. 167–176, Feb. 1999.

[29] D. Gamboa, C. Rego, and F. Glover, "Implementation analysis of efficient heuristic algorithms for the traveling salesman problem," *Comput. Oper. Res.*, vol. 33, no. 4, pp. 1154–1172, Apr. 2006.

[30] P. C. Gilmore, "Optimal and suboptimal algorithms for the quadratic assignment problem," *J. Soc. Ind. Appl. Math.*, vol. 10, no. 2, pp. 305–313, Jun. 1962.

[31] F. Glover, "A template for scatter search and path relinking," in *Lecture Notes in Computer Science*, vol. 1363, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, Eds. London, U.K.: Springer-Verlag, 1998, pp. 13–54.

[32] F. Glover, "Scatter search and path relinking," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. New York: McGraw-Hill, 1999, pp. 297–316.

[33] F. Glover, "Multi-start and strategic oscillation methods—Principles to exploit adaptive memory," in *Computing Tools for Modeling, Optimization, and Simulation: Interfaces in Computer Science and Operations*

Research, M. Laguna and J. L. Gozales Verlade, Eds. Boston, MA: Kluwer, 2000, pp. 1–24.

[34] F. Glover and M. Laguna, *Tabu Search*. Boston, MA: Kluwer, 1997.

[35] L. W. Hagen and A. B. Kahng, "Combining problem reduction and adaptive multistart: A new technique for superior iterative partitioning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 7, pp. 709–717, Jul. 1997.

[36] P. Hahn and J. Krarup, "A hospital facility layout problem finally solved," *J. Intell. Manuf.*, vol. 12, no. 5/6, pp. 487–496, Oct. 2001.

[37] M. Hasegawa, T. Ikeguchi, and K. Aihara, "Exponential and chaotic neurodynamical tabu searches for quadratic assignment problems," *Control Cybern.*, vol. 29, no. 3, pp. 773–788, 2000.

[38] R. Hübscher and F. Glover, "Applying tabu search with influential diversification to multiprocessor scheduling," *Comput. Oper. Res.*, vol. 21, no. 8, pp. 877–884, Oct. 1994.

[39] T. James, C. Rego, and F. Glover, "Sequential and parallel path-relinking algorithms for the quadratic assignment problem," *IEEE Intell. Syst.*, vol. 20, no. 4, pp. 58–65, Jul./Aug. 2005.

[40] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: A case study in local optimization," in *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J. K. Lenstra, Eds. New York: Wiley, 1997, pp. 215–310.

[41] P. Kadłuczka and K. Wala, "Tabu search and genetic algorithms for the generalized graph partitioning problem," *Control Cybern.*, vol. 24, no. 4, pp. 459–476, 1995.

[42] K. Katayama, M. Sadamatsu, and H. Narihisa, "Iterated k-opt local search for the maximum clique problem," in *Proc. EvoCOP*, 2007, vol. 4446, pp. 84–95.

[43] J. P. Kelly, M. Laguna, and F. Glover, "A study of diversification strategies for the quadratic assignment problem," *Comput. Oper. Res.*, vol. 21, no. 8, pp. 885–893, Oct. 1994.

[44] T. Koopmans and M. Beckmann, "Assignment problems and the location of economic activities," *Econometrica*, vol. 25, no. 1, pp. 53–76, 1957.

[45] G. Lan and G. DePuy, "On the effectiveness of incorporating randomness and memory into a multi-start metaheuristic with application to the set covering problem," *Comput. Ind. Eng.*, vol. 51, no. 3, pp. 362–374, Nov. 2006.

[46] M. Laguna, R. Marti, and V. Campos, "Intensification and diversification with elite tabu search solutions for the linear ordering problem," *Comput. Oper. Res.*, vol. 26, no. 12, pp. 1217–1230, Oct. 1999.

[47] E. L. Lawler, "The quadratic assignment problem," *Manage. Sci.*, vol. 9, no. 4, pp. 586–599, Jul. 1963.

[48] Y. Li, P. M. Pardalos, and M. G. C. Resende, "A greedy randomized adaptive search procedure for the quadratic assignment problem," in *Quadratic Assignment and Related Problems*, ser. DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 16, P. M. Pardalos and H. Wolkowicz, Eds. Baltimore, MD: Amer. Math. Soc., 1994, pp. 237–261.

[49] F.-T. Lin, C.-Y. Kao, and C.-C. Hsu, "Applying the genetic approach to simulated annealing in solving some NP-hard problems," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 6, pp. 1752–1767, Nov./Dec. 1993.

[50] H. R. Lourenco, O. C. Martin, and T. Stützle, "Iterated local search," in *Handbook of Metaheuristics*, ser. International Series in Operations Research and Management Sciences, vol. 57, F. Glover and G. Kochenberger, Eds. Boston, MA: Kluwer, 2003, pp. 321–353.

[51] V. Maniezzo and A. Colorni, "The ant system applied to the quadratic assignment problem," *IEEE Trans. Knowl. Data Eng.*, vol. 11, no. 5, pp. 769–778, Sep./Oct. 1999.

[52] O. Martin, S. W. Otto, and E. W. Felten, "Large-step Markov chains for the traveling salesman problem," *Complex Syst.*, vol. 5, no. 3, pp. 299–326, Jun. 1991.

[53] A. Marzetta and A. Brungger, "A dynamic-programming bound for the quadratic assignment problem," in *Proc. COCOON*, 1999, vol. 1627, pp. 339–348.

[54] T. Mautor and C. Roucairol, "A new exact algorithm for the solution of quadratic assignment problems," *Discrete Appl. Math.*, vol. 55, no. 3, pp. 281–293, Dec. 1994.

[55] A. Misevicius, "Genetic algorithm hybridized with ruin and recreate procedure: Application to the quadratic assignment problem," *Knowl.-Based Syst.*, vol. 16, no. 5/6, pp. 261–268, Jul. 2003.

[56] A. Misevicius, "An improved hybrid genetic algorithm: New results for the quadratic assignment problem," *Knowl.-Based Syst.*, vol. 17, no. 2–4, pp. 65–73, May 2004.

[57] A. Misevicius, "A tabu search algorithm for the quadratic assignment problem," *Comput. Optim. Appl.*, vol. 30, no. 1, pp. 95–111, Jan. 2005.

[58] V. Nissen and H. Paul, "A modification of threshold accepting and its application to the quadratic assignment problem," *OR Spektrum*, vol. 17, no. 2/3, pp. 205–210, Jun. 1995.

[59] V. Nissen, "Solving the quadratic assignment problem with clues from nature," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 66–72, Jan. 1994.

[60] M. Nystrom, "Solving certain large instances of the quadratic assignment problem: Steinberg's examples," California Inst. Technol., Pasadena, CA, Tech. Rep. CaltechCSTR:2001.010, 1999.

[61] C. A. Oliveira, P. M. Pardalos, and M. G. C. Resende, "GRASP with path-relinking for the quadratic assignment problem," in *Efficient and Experimental Algorithms*, vol. 3059, C. C. Ribeiro and S. L. Martins, Eds. London, U.K.: Springer-Verlag, 2004, pp. 356–368.

[62] G. Palubeckis, "Multistart tabu search strategies for the unconstrained binary quadratic optimization problem," *Ann. Oper. Res.*, vol. 131, no. 1–4, pp. 259–282, Oct. 2004.

[63] G. Palubeckis, "A multistart tabu search approach for graph coloring," *Inf. Technol. Control*, vol. 21, pp. 7–15, 2001.

[64] G. Palubeckis and V. Krivickiene, "Application of multistart tabu search to the MAX-CUT problem," *Informaacines Technologijos Ir Valdymas*, vol. 2, no. 31, pp. 29–35, 2004.

[65] P. M. Pardalos, L. S. Pitsoulis, and M. G. C. Resende, "A parallel GRASP implementation for the quadratic assignment problem," in *Parallel Algorithms for Irregular Structured Problems—Irregular'94*, A. Ferreira and J. Rolim, Eds. Boston, MA: Kluwer, 1995, pp. 111–130.

[66] P. M. Pardalos, F. Rendl, and H. Wolkowicz, "The quadratic assignment problem: A survey and recent developments," in *Quadratic Assignment and Related Problems*, ser. DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 16, P. M. Pardalos and H. Wolkowicz, Eds. Baltimore, MD: Amer. Math. Soc., 1994, pp. 1–42.

[67] J. W. Pepper, B. L. Golden, and E. A. Wasil, "Solving the traveling salesman problem with annealing-based heuristics: A computational study," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 32, no. 1, pp. 72–77, Jan. 2002.

[68] G. G. Polak, "On a special case of the quadratic assignment problem with an application to storage-and-retrieval devices," *Ann. Oper. Res.*, vol. 138, no. 1, pp. 223–233, Sep. 2005.

[69] M. Queyranne, "Performance ratio of heuristics for triangle inequality quadratic assignment problems," *Oper. Res. Lett.*, vol. 4, no. 5, pp. 231–234, Feb. 1986.

[70] C. Rego, "Relaxed tours and path ejections for the traveling salesman problem," *Eur. J. Oper. Res.*, vol. 106, no. 2/3, pp. 522–538, Apr. 1998.

[71] C. Rego, "RAMP: A new metaheuristic framework for combinatorial optimization," in *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, C. Rego and B. Alidaee, Eds. Boston, MA: Kluwer, 2005, pp. 441–460.

[72] C. Rego and R. Duarte, "A filter and fan approach to the job shop scheduling problem," *Eur. J. Oper. Res.*, vol. 194, no. 3, pp. 650–662, May 2009.

[73] C. Rego and F. Glover, "Local search and metaheuristics for the traveling salesman problem," in *The Traveling Salesman Problem and Its Variations*, G. Gutin and A. P. Punnen, Eds. Boston, MA: Kluwer, 2002, pp. 309–368.

[74] C. Rego, T. James, and F. Glover, "An ejection chain algorithm for the quadratic assignment problem," Univ. Mississippi, University, MS, Tech. Rep.

[75] S. Sahni and T. Gonzalez, "P-complete approximation problems," *J. Assoc. Comput. Mach.*, vol. 23, no. 3, pp. 555–565, Jul. 1976.

[76] P. D. Simic, "Constrained nets for graph matching and other quadratic assignment problems," *Neural Comput.*, vol. 3, no. 2, pp. 268–281, 1991.

[77] J. Skorin-Kapov, "Tabu search applied to the quadratic assignment problem," *ORSA J. Comput.*, vol. 2, no. 1, pp. 33–45, 1990.

[78] P. Soriano and M. Gendreau, "Diversification strategies in tabu search algorithms for the maximum clique problem," *Ann. Oper. Res.*, vol. 63, no. 2, pp. 189–207, Apr. 1996.

[79] T. Stützle, "Iterated local search for the quadratic assignment problem," *Eur. J. Oper. Res.*, vol. 174, no. 3, pp. 1519–1539, Nov. 2006.

[80] T. Stützle and M. Dorigo, "ACO algorithms for the quadratic assignment problem," in *New Ideas for Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. New York: McGraw-Hill, 1999, pp. 33–50.

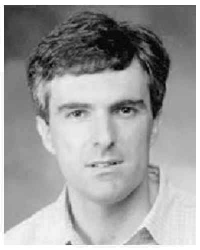
[81] E. Taillard, "Robust taboo search for the quadratic assignment problem," *Parallel Comput.*, vol. 17, pp. 443–455, 1991.

[82] D. M. Tate and A. E. Smith, "A genetic approach to the quadratic assignment problem," *Comput. Oper. Res.*, vol. 22, no. 1, pp. 73–83, Jan. 1995.

[83] L. Tseng and S. Liang, "A hybrid metaheuristic for the quadratic assignment problem," *Comput. Optim. Appl.*, vol. 34, no. 1, pp. 85–113, May 2006.
[84] M. R. Wilhelm and T. L. Ward, "Solving quadratic assignment problems by simulated annealing," *IIE Trans.*, vol. 19, no. 1, pp. 107–119, 1987.
[85] D. L. Woodruff, "Proposals for chunking and tabu search," *Eur. J. Oper. Res.*, vol. 106, no. 2/3, pp. 585–598, Apr. 1998.
[86] D. L. Woodruff and E. Zemel, "Hashing vectors for tabu search," *Ann. Oper. Res.*, vol. 41, no. 2, pp. 123–137, Jun. 1993.
[87] Q. Zhang, J. Sun, and E. Tsang, "An evolutionary algorithm with guided mutation for the maximum clique problem," *IEEE Trans. Evol. Comput.*, vol. 9, no. 2, pp. 192–200, Apr. 2005.

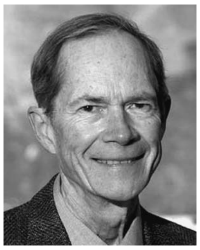
Tabitha James received the Ph.D. degree in business administration from the University of Mississippi, Oxford.

She is currently an Associate Professor with the Department of Business Information Technology, Pamplin College of Business, Virginia Polytechnic Institute and State University, Blacksburg. Her research interests include combinatorial optimization, heuristics, and parallel computing.



César Rego received the Ph.D. degree in computer science from the University of Versailles, Versailles, France.

He is currently an Associate Professor of management information systems and operations management with the School of Business, University of Mississippi, Oxford. His research focuses on creating and empirically validating optimization algorithms for solving complex and practical problems.



Fred Glover received the Ph.D. degree in operations research from Carnegie Mellon University, Pittsburgh, PA.

He is currently a Distinguished Professor with the University of Colorado at Boulder and the Chief Technology Officer with OptTek Systems, Inc. His research focuses on mathematical optimization, computer science, and artificial intelligence.

Dr. Glover is an Elected Member of the National Academy of Engineering. He is a recipient of the John von Neumann Theory Prize for lifetime contributions to operations research and management science.