An Ejection Chain Algorithm for the Quadratic Assignment Problem

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

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

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

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

Latest Revision: August 8, 2009.

Abstract – In this study we present a new tabu search algorithm for the quadratic assignment problem (QAP) that utilizes an embedded neighborhood construction called an ejection chain. Our ejection chain approach provides a combinatorial leverage effect, where the size of the neighborhood grows multiplicatively while the effort of finding a best move in the neighborhood grows only additively. Our results illustrate that significant improvement in solution quality is obtained in comparison to the traditional swap neighborhood. We also develop two multi-start tabu search algorithms utilizing the ejection chain approach in order to demonstrate the power of embedding this neighborhood construction within a more sophisticated heuristic framework. Comparisons to the best large neighborhood approaches from the literature are presented.

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

[1] Corresponding author.

1. Introduction

The quadratic assignment problem (QAP) is a classical combinatorial optimization problem that has garnered much attention due to both its large number of applications and its solution complexity. Originally used to model a location problem in the 1950's, the QAP is computationally very difficult to solve which makes it an ideal candidate for testing new algorithmic approaches. While facility location problems remain the most popular application area for the quadratic assignment problem, many other applications for this problem exist including scheduling problems, statistical data analysis, information retrieval, as well as problems in transportation. The attractiveness of the QAP is also due to the fact that many other combinatorial optimization problems can be formulated as a QAP, including the traveling salesman problem, the maximum clique problem and the graph partitioning problem. (See Cela (1998) for a survey of both classical and practical applications.)

In the context of facility location problems, the QAP can be stated as follows. Let $F = \{f_1,\ldots,f_n\}$ be a set of n facilities to be placed in exactly n locations represented by the set $L = \{l_1,\ldots,l_n\}$. $A = (a_{ik})$ is a matrix of *distances* between pairs of locations l_i, $l_k \in L$, and $B = (b_{jl})$ is an associated matrix of *flows* to be transmitted (or shipped) between pairs of facilities f_j, $f_l \in F$. The objective is to find a minimum cost assignment of facilities to locations considering both the flow of materials between facilities and the distance between locations.

In mathematical terms, each assignment can be defined as a permutation π of the underlying index set $N = \{1,\ldots,n\}$, i.e. $\pi : N \to N$. Hence, if facility j is assigned to location i and facility l is assigned to location k, the cost of the flow between facilities $j = \pi(i)$ and $l = \pi(k)$ is $a_{ik}b_{\pi(i)\pi(k)}$. The objective of the QAP is to find a permutation vector $\pi \in \Pi_n$ that minimizes the total assignment cost, where Π_n is the set of all possible permutations of N. Such a formulation can be generically described as

$$\underset{\pi\in\Pi_n}{Minimize} \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}b_{\pi(i)\pi(j)}.$$

Heuristic approaches for the QAP abound in the literature wherein local search is commonly used as a basic component to explore the solution space. Among these heuristics are tabu search (Taillard, 1991; Misevicius, 2005; James, Rego and Glover, 2009), scatter search (Cung et al., 1996), genetic algorithms (Fleurent and Ferland, 1994; Ahuja, Orlin and Tiwari, 2000; Misevicius, 2003, 2004; Drezner, 2003, 2005), ant colony optimization (Stutzle and Dorigo, 1999), GRASP (Li, Pardalos and Resende, 1994), GRASP with path relinking (Oliveira, Pardalos and Resende, 2004), and path relinking (James, Rego and Glover, 2005).

Local search methods rely on the exploration of a defined neighborhood to generate moves in the solution space of the problem under consideration. In the case of the QAP, this neighborhood is typically a 2-exchange neighborhood that swaps the location of two facilities at each step of the local search process. The exploration of larger neighborhoods where the simultaneous movement of k nodes of the permutation can be examined is attractive though computationally very demanding.

Ahuja et al. (2007) introduce a very large scale neighborhood search (VLSN) method for the QAP, which constitutes an important advance in the creation of more complex

neighborhoods for the problem. This algorithm iteratively examines all paths (or exchanges of nodes) of increasing depth, where the maximum depth is a specified parameter. The VLSN algorithm considers all moves (or a defined subset of moves) of a given depth before proceeding to the next depth. Due to the computational complexity of the full path enumeration scheme presented, a maximum path length of k=4 was settled upon in their study.

Ejection chain methods constitute a special class of very large neighborhoods that have proved highly promising in the solution of difficult and large scale combinatorial optimization problems. In general, ejection chains provide the ability to strategically extend simpler neighborhoods, such as those consisting of exchange (swap) moves or insert (shift) moves, to create more complex neighborhoods that can be generated with an efficient investment of effort (Glover, 1991). Some forms of ejection chain methods make use of a *reference structure* as a framework for generating moves at each level of the ejection chain construction (Glover, 1992, 1996).

Examples of successful applications of various types of ejection chains include: the multi-node insertion and exchange ejection chain method for the classical vehicle routing problem (Rego 2001), the long-chain shift neighborhood for the generalized assignment problem (Yagiura, Ibaraki and Glover, 2004), the stem-and-cycle (S&C) and the doubly-rooted S&C reference structures for the traveling salesman problem (Rego, 1998a, Rego et al. 2006), the flower reference structure for the vehicle routing problem (Rego, 1998b), and the subgraph ejection chain method for the crew scheduling problem (Cavique, Rego and Themido, 1999).

The key contribution of this paper is the development of a specialized ejection chain algorithm for the QAP, drawing on a proposal sketched in Glover (1991), which has useful features in the QAP setting. The approach utilizes the ejection chain structure to build successively larger exchanges based upon the elements chosen in the proceeding chain. In this manner, only a selected subset of all possible chains at each depth is considered for a given permutation. This process allows the method to quickly probe larger neighborhoods, with no constraints on the depths examined, by constructing these chains of moves based upon previously promising structures. More importantly, these ejection chain neighborhoods exhibit a special property called *combinatorial leverage*, where a level k neighborhood contains $\mathcal{O}(n^k)$ elements, but a potentially best member for a k-neighborhood $(k > 2)$ is determined with k examinations of $\mathcal{O}(n)$ "component" elements.

We embed our ejection chain method within a tabu search (TS) framework to provide strategic control over the formation of the chains. The first version of our TS method is extremely simple, using memory only in the role of "bookkeeping" operations instead of in the role of performing advanced guidance. Our chief purpose in examining this simple structure is to show that the ejection chain neighborhood obtains better solutions than an exchange neighborhood in the same framework. We then extend this basic framework to present two multi-start tabu search variants that yield solutions of higher quality and demonstrate the advantages of embedding ejection chains within a more sophisticated metaheuristic. We also provide computational comparisons to previous large neighborhood approaches.

2. The Ejection Chain Method

Our ejection chain method extends the classical 2-exchange (or swap) neighborhood for the QAP to effectively create more general *k*-exchange neighborhoods where *k* can take any integer value between 2 and *n*. The method may be conceived as providing a *variable depth neighborhood* that determines the value of *k* dynamically according to the current state of the search.

Underlying a general ejection chain design, exchange moves are successively embedded in the ejection chain construction, level by level, and are driven by the evaluation of two types of interrelated moves: (1) an *ejection move*, which extends the depth of the neighborhood by generating an intermediate (reference) structure; and (2) a *trial move*, which creates a feasible solution from the intermediate structure provided by the ejection move. The structure obtained with the application of the trial move is called a *trial solution*.

Our QAP ejection chain method constitutes a node-based ejection chain model where facilities are associated with nodes in a graph which are to be assigned to locations. In this context the method implements a type of *multi-node exchange move*, which can be seen as a series of swap moves for the QAP.

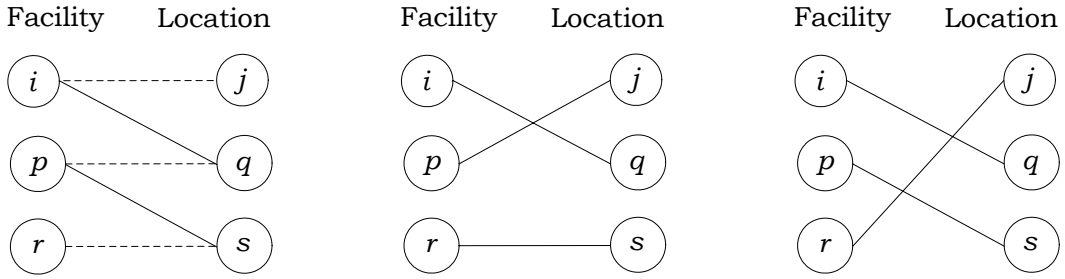
2.1 The Ejection Chain Neighborhood

We represent a QAP solution as a *perfect matching in a bipartite graph*. Let $G = (F \cup L, F \times L)$ be a (complete) bipartite graph with $F = \{f_1, \ldots, f_n\}$ representing facilities and $L = \{l_1, \ldots, l_n\}$ representing locations. A solution for the QAP can be defined by a partial graph $S = (F \cup L, E \subset F \times L)$ such that $(f_i, l_j) \in E$ if and only if facility f_i is assigned to location l_j and no two arcs are incident to the same node.

An ejection chain neighborhood can be defined on a subgraph $H = (W, T)$ of S where $T = \{(f^0, l^0), \ldots, (f^k, l^k), \ldots (f^l, l^l)\}$ is a set of arcs representing $l+1$ levels of an ejection chain, which we denote by $T = \bigcup_{k=0}^{l} \{(f^k, l^k)\}$. An *ejection* results by moving a facility f_i from location l_j to a new location l_q occupied by another facility f_p, disconnecting f_p from its location. In terms of the aforementioned graph formulation, this move is equivalent to deleting arcs (f_i, l_j), (f_p, l_q), and inserting an arc (f_i, l_q). Let k be a level of the chain, each node f^k ejects the node f^{k+1} ending with the ejection of the node f^l. As a result, an ejection chain of $l+1$ levels is the replacement of T by $T' = \bigcup_{k=1}^{l} \{(f^{k-1}, l^k)\}$, transforming S into a disconnected graph. In other words, arcs $(f^k, l^k)\,(k = 0, \ldots, l)$ are successively replaced by arcs $(f^{k-1}, l^k)\,(k = 1, \ldots, l)$. Because f^l is not assigned to any location, this transformation does not represent a complete transition from the current solution S to a new feasible solution S'. However, the complete transition can be obtained by a trial move that connects the graph by simply inserting the arc (f^l, l^0). Let T'' be the set defined by the arc added by the trial move, the new neighboring solution is obtained as $S' = S \cup T' \cup T'' - T$.

The general model is illustrated in Figure 1 for three levels (0, 1, and 2) of an ejection chain. Diagram A depicts the ejection moves performed throughout the ejection chain, and diagrams B and C illustrate the connected graphs obtained by the trial moves at

levels 1 and 2, respectively. Dotted lines represent the set T associated with original assignments in the solution S that were affected by the ejection chain process. Likewise, solid lines denote the sets T' and T'' representing the new assignments made by the ejection moves and the associated trial moves. Specifically, for level 1, we have $T = \{(i, j), (p, q)\}$, $T' = \{(i, q)\}$, and $T'' = \{(p, j)\}$. By extension, for level 2, we have $T = \{(i, j), (p, q), (r, s)\}$, $T' = \{(i, q), (p, s)\}$, and $T'' = \{(r, j)\}$.



A. Ejection Moves B. Trial Move at Level 1 C. Trial Move at Level 2

Figure 1–Illustration of two levels of an ejection chain for the QAP

The process continues through additional nodes of G until a suitable termination criterion is met. The adaptation of the ejection chain idea to this setting may be viewed as a generalization of a weighted alternating path approach, as applied in the solution of matching problems.

2.2 The Ejection Chain Construction

The evaluation of moves is a critical factor in building an ejection chain. Handled appropriately, the evaluation of ejection and trial moves yields an important form of *combinatorial leverage* in the creation of k-exchange neighborhoods of the type exploited in this study. In our construction, the number of moves represented by a level k neighborhood is multiplicatively greater than the number of moves in a level k-1 neighborhood, but the best move from the neighborhoods at each individual level $(k > 1)$ can be determined by adding only the effort required to examine the neighborhood of a single node. In particular, the number of moves composing the first, second, and third levels are $\mathcal{O}(n^2)$, $\mathcal{O}(n^3)$, and $\mathcal{O}(n^4)$, but the best member of level two and three neighborhoods can be found by adding only $\mathcal{O}(n)$ effort to the work expended to determine the best first level move. The method is based on the principle of capturing relevant component moves in successive neighborhoods as a way to generate good compound moves—potentially the best in the associated k-level neighborhood.

To understand the operation of these moves, consider starting with a simple 2-exchange neighborhood. If the best 2-exchange is not improving, there may be a sequence of moves going beyond 2-exchanges that can do better. For instance, the compound move exemplified in the diagram C of Figure 1 corresponds to two successive 2-exchange moves, where facility $i = \pi(j)$ is first exchanged with facility $p = \pi(q)$, and then $p = \pi(j)$ is exchanged with $r = \pi(s)$. Since location j is involved in both 2-exchange moves, this neighborhood implements a 3-exchange move. Accordingly, a $k - 1$ level ejection chain neighborhood of this type is shown to implement general k-exchange moves. It follows

that the second level neighborhood contains $\mathcal{O}(n^3)$ moves (barring the use of candidate lists) since each of the n choices for i can eject $\mathcal{O}(n)$ alternatives for node p, which in turn can eject $\mathcal{O}(n)$ other alternatives. However, we can identify the best move from a closely related $\mathcal{O}(n^3)$ neighborhood by one application of $\mathcal{O}(n^2)$ effort and one of $\mathcal{O}(n)$ effort. The $\mathcal{O}(n^3)$ neighborhood we treat is actually less encompassing than the one indicated, as a result of a construction that avoids duplications among certain nodes at different levels to insure the legitimacy of the compound moves ultimately produced. As long as the number of levels is small relative to n, the combinatorial leverage is not significantly affected by this legitimacy-preserving construction. On the other hand, there can be advantages to extending the number of levels for the purpose of inducing a diversification effect to overcome local optimality.

To evaluate the change in solution cost created by a compound move at a given level k of an ejection chain, it is convenient to subdivide these changes into two fundamental component operations: *disconnecting* the facility currently assigned to location j, and *relocating* facility i to location j. Denote the first ejected node (which initiates the chain) by the *top* node t, and the current ejected node by the *bottom* node b. We let $\pi(i)$ represent the facility at location i in a solution corresponding to a trial ejection chain under consideration, and $\pi'(i)$ represent the facility at location i in a current solution.

Because the selection of the initial top and bottom nodes requires the evaluation of the trial move that is made after ejecting the potential bottom node, the relocation of the bottom node into the position vacated by the top node must be evaluated before relocating the top node. This particularity makes the relocation operation at the first level of the ejection chain different from the relocations used in the ejection and trial moves performed at higher levels of the chain (where the relocation of the current bottom node is evaluated after the bottom node at the previous level already occupies its new position).

For this reason, it is convenient to define a special relocation operation aimed at *circularizing* the ejection move at the first level of the ejection chain. The cost changes associated with these operations may be expressed as follows:

Disconnection value: $$\mathcal{D}(j) = -\sum_{h=1}^{n} a_{hj} b_{\pi(h)\pi(j)} \qquad h \neq j, t$$

Relocation value: $$\mathcal{R}(i, j) = \sum_{h=1}^{n} a_{jh} b_{i\pi(h)} \qquad h \neq j, t$$

Circularization value: $$\mathcal{C}(j) = \begin{cases} \sum_{h=1}^{n} a_{th} b_{\pi(j)\pi(h)} & h \neq j, t \\ \sum_{h=1}^{n} a_{th} b_{\pi(t)\pi(h)} & h = j, h \neq t \end{cases}$$

Hence, for the symmetric QAP, the actual solution cost change associated with these operations is twice the value obtained by the corresponding operation. The generalization of these operations to the asymmetric variant of the problem can be

obtained by simply creating additional product terms that switch the indexes of the product terms above and adding these new terms to the preceding expressions.

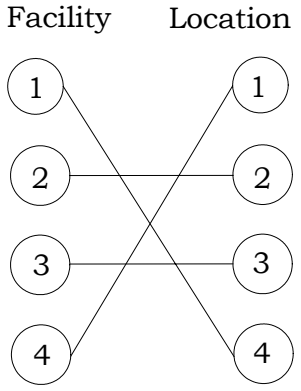
Let *ejection value* denote the solution cost change associated with the ejection moves, and let *trial value* denote the solution cost change associated with a trial move. Then, an ejection chain of l levels satisfying the requirements of legitimacy may be recursively evaluated as follows:

$$\textit{Ejection value}: \quad \mathcal{E}(k) = \begin{cases} \mathcal{D}(t) + \mathcal{D}(b^k) + \mathcal{R}(\pi(t), b^k) & k = 1 \\ \mathcal{E}(k-1) + \mathcal{D}(b^k) + \mathcal{R}(\pi'(b^{k-1}), b^k) & 1 < k \le l \end{cases}$$

$$\textit{Trial value}: \quad \Delta(k) = \begin{cases} \mathcal{E}(k) + C(b^k) & k = 1 \\ \mathcal{E}(k) + \mathcal{R}(\pi'(b^k), t) & 1 < k \le l \end{cases}$$

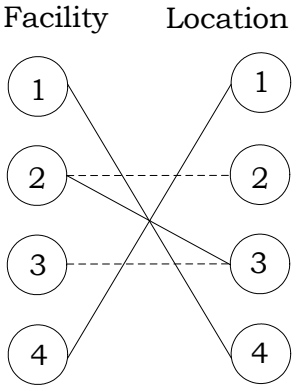
Letting $Z(S)$ be the cost of the current QAP solution S, the value of a trial solution S^k obtained at a level k of the ejection chain is given by $Z(S^k) = Z(S) + 2\,\Delta(k)$ for the symmetric case. As previously mentioned, for the asymmetric case, the last term would include the reverse products in $\mathcal{D}$, $\mathcal{R}$, and $\mathcal{C}$ rather than being doubled. The method keeps track of the level k^* where the best trial solution has been found, which corresponds to the depth of the compound move applied to the current solution S so as to obtain the new neighboring solution S'. Figure 2 gives an example of these calculations for the evaluation of two levels of an ejection chain. To simplify the illustration and keep the equations short, the example considers the symmetric QAP and assumes that the transition to a new neighboring solution is performed at level two.

Illustrative example:



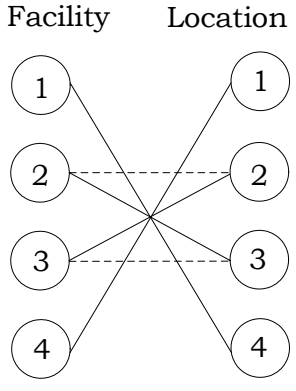
Starting Solution S

$$S = \langle 4, 2, 3, 1 \rangle$$



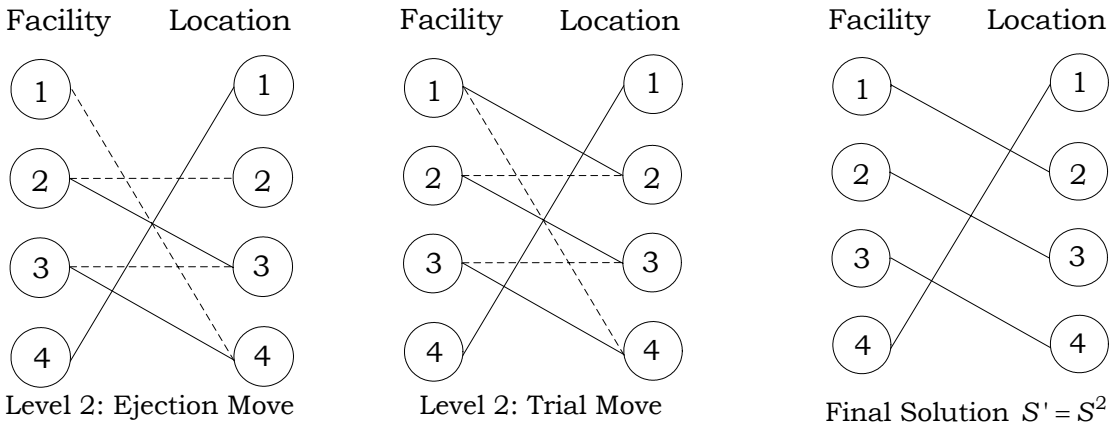
Level 1: Ejection Move

$$\mathcal{D}(2) = -a_{12}b_{24} - a_{23}b_{23} - a_{24}b_{12}$$
$$\mathcal{D}(3) = -a_{13}b_{34} - a_{34}b_{13}$$
$$\mathcal{R}(2,3) = a_{31}b_{24} + a_{34}b_{12}$$
$$\mathcal{E}(1) = \mathcal{D}(2) + \mathcal{D}(3) + \mathcal{R}(2,3)$$



Level 1: Trial Move

$$\mathcal{C}(3) = a_{12}b_{34} + a_{23}b_{23} + a_{24}b_{13}$$
$$\Delta(1) = \mathcal{E}(1) + \mathcal{C}(3)$$
$$S^1 = \langle 4, 3, 2, 1 \rangle$$
$$Z(S^1) = Z(S) + 2\,\Delta(1)$$



$$\mathcal{D}(4) = -a_{14}b_{14} - a_{34}b_{12}$$
$$\mathcal{R}(3,4) = a_{14}b_{34} + a_{34}b_{23}$$
$$\mathcal{E}(2) = \mathcal{E}(1) + \mathcal{D}(4) + \mathcal{R}(3,4)$$

$$\mathcal{R}(1,2) = a_{12}b_{14} + a_{23}b_{12} + a_{24}b_{13}$$
$$\Delta(2) = \mathcal{E}(2) + \mathcal{R}(1,2)$$
$$S^2 = \langle 4,1,2,3 \rangle$$
$$Z(S^2) = Z(S) + 2\,\Delta(2)$$

Figure 2–Evaluation of an ejection chain of two levels

In the illustration, the chain starts with $t = 2$ and $b^1 = 3$ as the initial top and bottom nodes, respectively. The first operations consist of disconnecting these two nodes from the graph and relocating (facility) node 2 in the location previously occupied by node 3, keeping node 2 disconnected. The algebraic sum of these three operations gives the value of the ejection move $\mathcal{E}(1)$ for the first level of the chain. The value of the trial move $\Delta(1)$ associated with the current ejection is obtained by circularizing the chain, relocating the current ejected node 3 to occupy the location vacated by the top node 2. At this point, the value of the corresponding trial solution $Z(S^1)$ can be calculated by adding the circularization value to the value of the starting solution. The second level is created by choosing facility 1 at location 4 to be ejected by the currently disconnected (bottom) node $b^1 = 3$, thus setting $b^2 = 4$. The new ejection value is then computed by adding the disconnection and relocation values $\mathcal{D}(4)$ and $\mathcal{R}(3,4)$ of the current ejection to the previously obtained ejection value $\mathcal{E}(1)$. Finally, the new trial value $\Delta(2)$ is obtained by adding the relocation value of facility 1 into the original position of the top node to the current ejection value.

2.2 The Ejection Chain Procedure

The ejection chain method begins by identifying the best local move for each facility *j*, which constitutes removing *j* from its current location and relocating it in the position occupied by a facility *l*, which is thereby ejected. (The method can also start by looking at each *l* and finding the best *j* to replace it.) The first level of the ejection chain consists of selecting initial chains based on performing a series of best 2-exchange moves. Notably, such a move corresponds to simultaneously determining the best initial node to be ejected and the best node to occupy the location of the ejected node. The chain grows by selecting a new node to be ejected by the previously ejected node. Under the natural restriction that prevents an element from being moved twice, the chain can continue to grow until all *n* nodes have been ejected. The pseudocode for the ejection chain procedure is sketched in Figure 3.

Step 0. Initialization

(a) Let π be the permutation associated with a starting solution S.

(b) Set k=1, $e^k = \varnothing$.

Step 1. Create the first level of the ejection chain

(a) Determine elements t and b^k that produce the lowest trial value $\Delta(k)$ over all elements $t, b^k \in N;\ t \neq b^k$. Let $\mathcal{E}(k)$ be the corresponding ejection value.

(b) Set $e^{k-1} = \{t\},\ e^k = e^{k-1} \cup \{b^k\}$. Let $k^* = k$ be the current best level.

(c) Update current trial solution by setting $\pi(t) = \pi(b^k)$ and $\pi(b^k) = \pi'(t)$.

(d) If $k = L$ go to Step 3. Otherwise go to Step 2.

Step 2. Grow the chain to further levels

(a) Set k=k+1.

(b) Determine the new element b^k that minimizes the ejection value $\mathcal{E}(k)$ over all elements $b^k \in N;\ b^k \notin e^k$.

(c) Set $e^k = e^{k-1} \cup \{b^k\}$.

(d) Update current trial solution by setting $\pi(t) = \pi(b^k)$ and $\pi(b^k) = b^{k-1}$.

(e) Compute trial value $\Delta(k)$ for the current level k.

(f) Keep track of the best level k* that produced the best trial value.

(g) If k<L and $|e^k| < n$ return to Step 2(a). Otherwise go to Step 3.

Step 3. Perform the compound move

(a) Apply to S the sequence of ejection moves in e^{k*}, recursively.

(b) Perform the trial move on S for the level k* to create a new solution S'.

(c) Compute the cost $Z(S')$ of the new current solution.

Figure 3–The ejection chain procedure

3. Tabu Search Algorithms

Rudimentary tabu search (TS) approaches of the type considered here employ short term memory structures to forbid moves that lead to solutions recently visited (rendering these moves *tabu).* One or more aspiration criteria are typically employed that allows the tabu status of a move to be overridden when the move exhibits desirable characteristics. More advanced TS implementations include the use of long term memory to restrict or encourage moves based on frequency and logical analysis, and incorporate intensification and diversification strategies to encourage the search towards promising and unexplored regions of the search space, respectively. For a comprehensive treatment of TS, see Glover and Laguna (1997).

The first TS approach we consider, denoted EC1, is used only to provide a comparison between different neighborhoods, and minimizes the TS mechanisms employed. EC1 uses a tabu restriction that renders moves tabu for only a short period and is used to compare the classical swap neighborhood to the ejection chain neighborhood. We then develop two additional tabu search algorithms denoted EC2 and EC3, using a multi-start design to provide a basic form of diversification. While still utilizing only simple TS strategies, these algorithms illustrate the potential of the ejection chain approach when embedded within a slightly more advanced framework.

3.1 The basic ejection chain algorithm: EC1

Starting from a randomly generated initial permutation, the EC1 tabu search algorithm utilizes a tabu list to restrict only the choice of the initial top and bottom nodes of the ejection chain construction. Once the initial nodes of a chain are selected, a *tabu tenure* is chosen for each of the two nodes that determines the number of subsequent iterations in which these nodes are tabu, meaning in this case that they are prevented from starting another chain. Since these tabu restrictions only apply to the two initial nodes, associated checking and updating of the tabu list are implemented in Step 1(a) of the ejection chain procedure of Figure 3. An aspiration criterion is not used in EC1 to override the tabu restrictions, since we give them a very small tenure. To obtain a direct comparison of the neighborhoods, a 2-exchange (or swap) neighborhood version of EC1 was also implemented by restricting the chain length to two nodes, i.e. $L = 1$. The EC1 algorithm imposes a minimum amount of heuristic guidance on the search and illuminates the impact of the neighborhood definition utilized. As a basis for this we keep track of the number of consecutive iterations with no improvement of the global best solution (NF) and stop the algorithm when this counter reaches a predefined maximum number of failures (MF). The basic algorithm is shown in Figure 4.

3.2 Multi-start ejection chain algorithms: EC2 and EC3

Multi-start algorithms seek to perturb the standard search path by periodically re-launching the search from a new initial configuration. A multi-start tabu search for the QAP is given by Fleurent & Glover (1999) where a local search is iteratively applied to solutions built by a constructive method tailored to provide high quality starting solutions. Another approach for diversifying the solutions generated is to make parameter adjustments to influence the trajectory of the search. While such an approach is not a multi-start approach in the classical sense, it likewise leads to a new solution that may be interpreted as a new starting point for the search, and hence for convenience we will refer to it as a multi-start procedure in the discussions of this paper. The multi-start procedures introduced in the current study are of both types.

EC2 and EC3 differ in the solution that ultimately replaces the current working permutation when the algorithm is restarted. Both algorithms impose a simple tabu restriction on the initial nodes chosen as considered in EC1. However, for these multi-start variants the tabu tenure is increased and an aspiration criterion is applied that allows a tabu move to be made under certain conditions, as follows. First, we apply the aspiration criterion only if the previous iteration of the local search did not produce a globally improving solution. Next, the move must meet two conditions: (1) the cost of the move must be less than that of the best move found so far during the current iteration; (2) a move meeting the first condition is permissible if the tabu tenure of the elements of the restricted move fall below a predefined aspiration threshold. Since these conditions restrict the choices of the two initial nodes used to start the chain, they are tested in Step 1(a) of the ejection chain procedure in Figure 3.

The number of non-improving iterations (or failures) since the last perturbation is kept by the NRF counter. Both variants are restarted when an improving solution is not found within a predetermined number of iterations since the last perturbation was applied. The maximum restart failures threshold value MRF is drawn from a range determined by the stopping criterion parameter. At each restart, this value is redrawn to allow the search stagnation threshold to vary within a controlled range throughout the run of the algorithm. For both EC2 and EC3, when the maximum restart failures

threshold is reached (*NRF* = *MRF*), the tabu parameters are reset to change the trajectory of the search.

In EC2 the current working solution is replaced by the global best solution from the previous iterations of the search. EC3 restarts from a diversified version of the best solution. In this variant, a diversification method is applied to a copy of the best permutation and the current working solution is replaced with this diversified permutation. Figure 5 provides the pseudocode for the general multi-start algorithm.

Step 0. Initialization
- (a) Generate a starting solution S.
- (b) Let $S^* = S$ be the current best solution.
- (c) Input parameters: maximum number of failures MF and maximum levels L.

Step 1. Perform tabu search
- (a) Call the ejection chain procedure with parameters S and L
- (b) Set $S = S'$.
- (c) If strictly improving:
  - (c1) Update the best solution with the new current solution, $S^* = S$.
  - (c2) Set number of failures $NF = 0$.
- (d) Otherwise, set $NF = NF + 1$.
- (e) If $NF < MF$ return to Step 1(a). Otherwise stop.

Figure 4–Simple tabu tenure ejection chain algorithm: EC1

Step 0. Initialization
- (a) Generate a starting solution S.
- (b) Let $S^* = S$ be the current best solution.
- (c) Input parameters: maximum number of failures MF and maximum levels L.

Step 1. Perform tabu search
- (a) Call the ejection chain procedure with parameters S and L
- (b) Set $S = S'$.
- (c) If strictly improving:
  - (c1) Update the best solution with the new current solution, $S^* = S$.
  - (c2) Set number of failures $NF = 0$.
  - (c3) Disallow aspiration.
- (d) Otherwise:
  - (d1) Increment number of failures $NF = NF + 1$.
  - (d2) Increment number of restart failures $NRF = NRF + 1$.
  - (d3) Allow aspiration.
  - (d4) If $NRF > MRF$ (number restart failures > maximum restart failures)
    - (d4.1) Update aspiration threshold.
    - (d4.2) Remove tabu restrictions for all elements.
    - (d4.3) Modify tabu tenure.
    - (d4.4) Set $NRF = 0$ and modify MRF value.
    - (d4.5) Set current solution $S = S^*$.
    - (d4.6) (If EC3) diversify (best solution) S.
- (e) If $NF < MF$ return to Step 1(a). Otherwise stop.

Figure 5–Multi-start ejection chain algorithm variants: EC2 and EC3

The diversification procedure used in EC3 was suggested by Glover (1998) and its pseudocode is given in Figure 6. This method creates a new solution from a seed solution (in this case the current global best permutation) by defining a step size and then reordering the permutation based upon this step size. Starting from a step size of 2, the step size is increased each time the algorithm is restarted, cycling back to the original step size if necessary.

Step 0. Initialization
- (a) Consider π be a permutation associated with an input solution S
- (b) Consider a seed permutation φ and set $\varphi = \pi$.
- (c) Set $k = 1$.
- (d) Set $start = step$ and set $j = start$.

Step 1. Generate diverse solution
- (a) Set $\pi(k) = \varphi(j)$.
- (b) Set $k = k + 1$.
- (c) If $j < n$ set $j = j + step$ and go to Step 1(a).
- (d) If $start > 1$ set $start = start + 1$ and to Step 1(a). Otherwise stop.

Figure 6–Diversification procedure

To illustrate the method given in Figure 6, consider the following example permutation:

$$\varphi = \langle 3, 5, 8, 1, 4, 6, 2, 7 \rangle.$$

Choosing a step size of 2, the first iteration of the algorithm initializes the *start* variable to 2, which causes *j* to range from 2 to the number of elements in the permutation, in this example n = 8. After the first iteration of the outer loop, the following partial permutation is obtained:

$$\pi = \langle 5, 1, 6, 7, \_, \_, \_, \_ \rangle.$$

The next pass through the outer loop then sets $start = 1$ and the following complete permutation is obtained:

$$\pi = \langle 5, 1, 6, 7, 3, 8, 4, 2 \rangle.$$

In this manner, for each step size a different permutation is obtained.

4. Computational Results

All algorithms were tested on a standard set of QAP benchmark instances obtained from QAPLIB (Burkard et al., 1997). All algorithms were written in the C programming language and run on a single Intel Itanium processor (1.3 GHz) on a SGI Altix running the Linux operating system. The parameters for each algorithm variant (EC1, EC2, and EC3) developed in this study are summarized in Table 1.

Parameter	EC1	EC2	EC3
Maximum Failures (MF) (Stopping Criterion SC1)	$5000n$	5000n	5000n
Time Limit (Stopping Criterion SC2)	1 hour ($n \leq 40$) 2 hours ($n > 40$)	1 hour ($n \leq 40$) 2 hours ($n > 40$)	1 hour ($n \leq 40$) 2 hours($n > 40$)
Allowable Failures (MRF)			
Lower Limit	n/a	$5n$	$5n$
Upper Limit	n/a	$500n$	$500n$
(Restart Criterion)			
Tabu Tenure			
Lower Limit (LT)	3 (static)	$n/10$ (variable)	$n/10$ (variable)
Upper Limit (UT)	10 (static)	$3n/10$ (variable)	$3n/10$ (variable)
Aspiration Threshold	n/a	(LT+UT)/2	(LT+UT)/2
Restart Tabu Tenure			
Lower Limit (LT)	n/a	$n/10$	$n/10$
Upper Limit (UT)	n/a	n	n
Restart Solution	n/a	global best	diversified global best

Table 1–TS Variant Parameter Settings

We consider runs under two stopping conditions, denoted by SC1 and SC2. SC1 caused the algorithms to cease execution of the search after no improvement is found in $5000n$ iterations (*MF*), where n is the number of facilities/locations, or the problem size. SC2 stipulates a time limit of 1 hour for instances of size $n \leq 40$, and 2 hours for larger instances, after which the algorithm terminates execution. SC2 is the same stopping condition applied in Ahuja et al. (2007) and is used to allow for a direct comparison with the associated VLSN algorithms.

Tables 2 and 3 present computational results for all variants of the algorithms under SC1 and SC2 stopping conditions, respectively. The parameters used in all algorithms for both Tables 2 and 3 are the same with the exception of the stopping condition. All algorithms ran 10 times on each problem instance, each time starting from a randomly generated seed solution. The tabu tenure for EC1 was set to be an integer value randomly drawn from the range 3 to 10. The tabu tenure for EC2 and EC3 is initialized with a value chosen from the range $n/10$ and $3n/10$. At each restart for EC2 and EC3, the tabu search parameters are adjusted. The upper and lower limits, that are used to determine the tabu tenure for an element are redrawn and allowed to vary in the range $n/10$ to n. Similarly, the maximum restart failures (*MRF*) parameter is reset every time a restart occurs for both the EC2 and EC3 variants. At each restart *MRF* is chosen from the range $5n$ to $500n$. As previously mentioned no aspiration criterion is used in EC1, while for EC2 and EC3 tabu active moves are subjected to two aspiration conditions. In our implementation, the required aspiration threshold is defined to be the average of the lower and upper limits of the current tabu tenure range. Also, as remarked EC2 and EC3 algorithms differ in the mechanism used to restart the search: EC2 restarting from the current global best solution, and EC3 restarting from a diversified version of the global best solution.

Table 2 and Table 3 follow the same format. The first two columns provide the name of the test instance and the corresponding best known solution (BKS). The next columns are organized in four groups associated with each variant of our ejection chain algorithm. The first two groups represent the simple tabu search algorithm restricted to first-level ejection chains to implement a 2-exchange neighborhood (2-exchange EC1), and its extension to *n*-level ejection chains implementing a variable depth *k*-exchange neighborhood (EC1). The next two groups correspond to the two multi-start tabu search variants using either the current global best solution (EC2) or a diversified version of it (EC3) as a perturbation scheme to restart the search. For each algorithm we provide the average percent deviation (APD) to the BKS, the best percent deviation (BPD) for only the best solution obtained from the 10 runs, the average iteration the best solution was found (ABI), and either the average running time to completion (ATTC) in minutes using stopping criterion SC1 (Table 2) or the average running time to solution (ATTS) using stopping criterion SC2 (Table 3).

Figure 7 graphically depicts the average solution quality of the three EC variants and the 2-exchange neighborhood from Table 2. Figure 8 shows a comparison of the average times to completion for all algorithms in Table 2 on each problem instance.

		2-Opt EC1				EC1				EC2				EC3			
Problem	BKS	APD	BPD	ABI	ATTC	APD	BPD	ABI	ATTC	APD	BPD	ABI	ATTC	APD	BPD	ABI	ATTC
Skorin-Kapov Instances																	
sko42	15812	0.120	0.000	121327	2.46	**0.028**	0.000	141003	4.78	0.293	0.000	127261	6.16	0.061	0.000	181144	7.15
sko49	23386	0.199	0.051	81096	4.51	0.199	0.000	56821	7.05	0.235	0.051	105762	11.00	**0.086**	0.051	164692	13.02
sko56	34458	0.418	0.012	101949	8.70	0.527	0.075	69944	12.78	0.475	0.012	204842	22.96	**0.259**	0.029	144841	19.97
sko64	48498	0.644	0.107	145538	19.66	0.464	0.000	86411	27.64	0.243	0.008	271342	50.46	**0.139**	0.000	330584	55.50
sko72	66256	0.932	0.211	211365	34.30	0.691	0.072	224179	54.37	**0.322**	0.060	252483	69.86	0.340	0.211	310567	76.20
sko81	90998	0.901	0.090	217687	58.79	0.849	0.259	76567	70.54	0.336	0.097	299275	121.94	**0.271**	0.044	378543	136.34
sko90	115534	0.691	0.230	254279	97.54	0.881	0.389	104395	117.12	0.305	0.000	587829	256.00	**0.272**	0.014	443539	219.75
sko100a	152002	0.736	0.378	225563	146.47	0.661	0.404	182165	213.58	0.314	0.026	444283	334.82	**0.263**	0.133	451170	337.15
sko100b	153890	0.868	0.659	176445	136.42	0.803	0.298	338037	262.14	0.379	0.127	379273	311.93	**0.226**	0.087	378051	311.29
sko100c	147862	1.108	0.557	256918	152.55	0.779	0.170	176688	211.84	0.731	0.030	616575	396.26	**0.269**	0.107	470522	344.28
sko100d	149576	1.007	0.389	59199	112.66	0.862	0.594	106422	189.76	0.437	0.070	369067	307.84	**0.316**	0.060	667656	414.04
sko100e	149150	1.178	0.801	363182	174.00	1.147	0.615	188224	215.46	0.522	0.020	428918	329.18	**0.198**	0.020	583862	384.48
sko100f	149036	0.989	0.769	324463	166.58	0.979	0.498	76796	180.70	0.567	0.169	468075	344.04	**0.395**	0.203	412134	323.81
Average		0.753	0.327	195308	85.74	0.628	0.260	140589	120.60	0.397	0.052	350383	197.11	**0.238**	0.074	378254	203.31
Symmetric Taillard Instances																	
tai20a	122455319	0.906	0.000	70967	0.38	0.718	0.304	60805	0.31	**0.152**	0.000	66085	0.22	0.199	0.000	50303	0.20
tai25a	344355646	1.523	0.937	109665	0.23	0.670	0.000	63245	0.27	0.294	0.000	61480	0.71	**0.055**	0.000	76388	0.69
tai30a	637117113	0.959	0.398	144811	0.45	0.645	0.490	127572	1.29	0.178	0.000	71869	1.60	**0.137**	0.000	61311	1.29
tai35a	283315445	1.123	0.595	143038	1.08	0.972	0.698	98117	1.92	0.302	0.000	143831	3.29	**0.272**	0.000	151137	3.38
tai40a	637250948	1.022	0.439	172807	2.19	0.825	0.416	202577	4.50	0.420	0.305	139842	5.38	**0.387**	0.120	183452	6.10
tai50a	458821517	1.090	0.710	212317	6.67	1.058	0.721	198235	11.14	0.732	0.572	169572	13.81	**0.726**	0.564	189709	14.30
tai60a	608215054	1.025	0.775	230485	16.99	1.017	0.602	157755	23.78	**0.715**	0.499	308515	40.43	0.861	0.601	134047	28.60
tai80a	818415043	0.804	0.625	324638	64.78	0.695	0.453	311313	97.93	**0.644**	0.392	465271	142.07	0.780	0.581	211760	100.09
tai100a	1185996137	**0.481**	0.348	491175	200.14	0.563	0.311	304380	252.10	0.600	0.272	332753	295.57	0.654	0.419	321325	291.44
Average		0.993	0.536	211100	32.55	0.796	0.444	169333	43.69	**0.449**	0.227	195469	55.90	0.452	0.254	153270	49.56
Overall		0.851	0.413	201796	63.98	0.729	0.335	152348	89.13	0.418	0.123	287009	139.34	**0.326**	0.147	286215	140.41

Table 2–Computational results for Skorin-Kapov problems and symmetric Taillard problems using Stopping Criterion 1

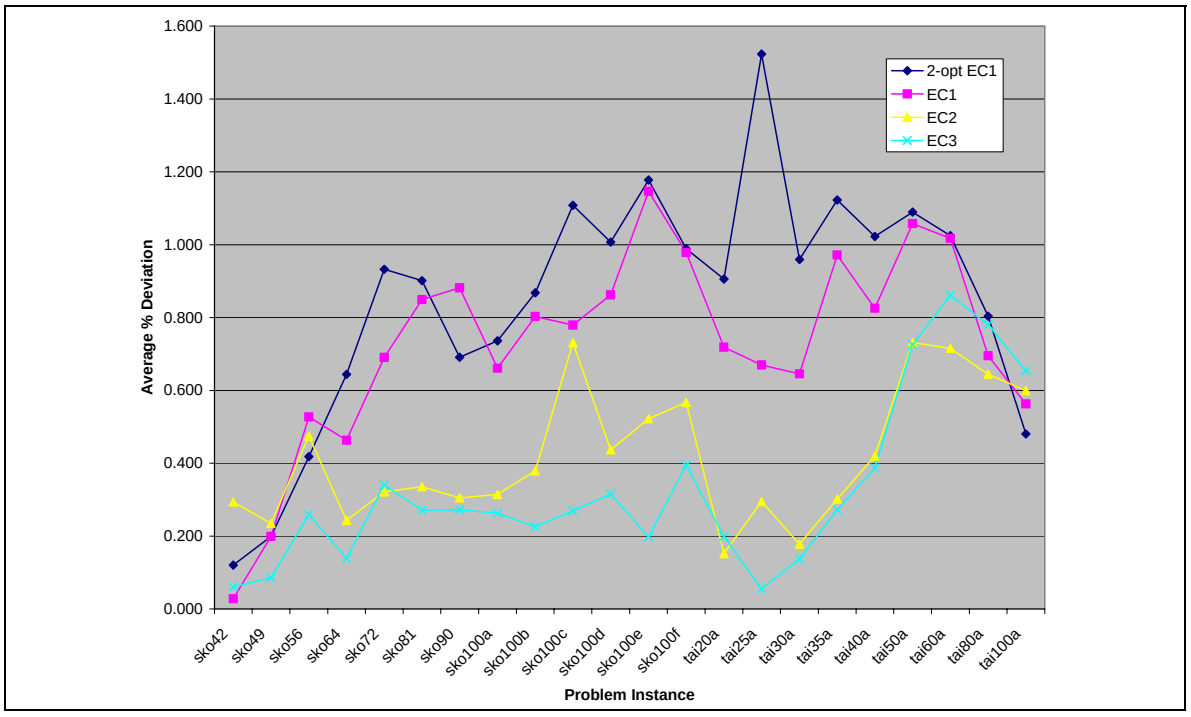


Figure 7–Average Percent Deviation (APD) for Skorin-Kapov and symmetric Taillard Instances

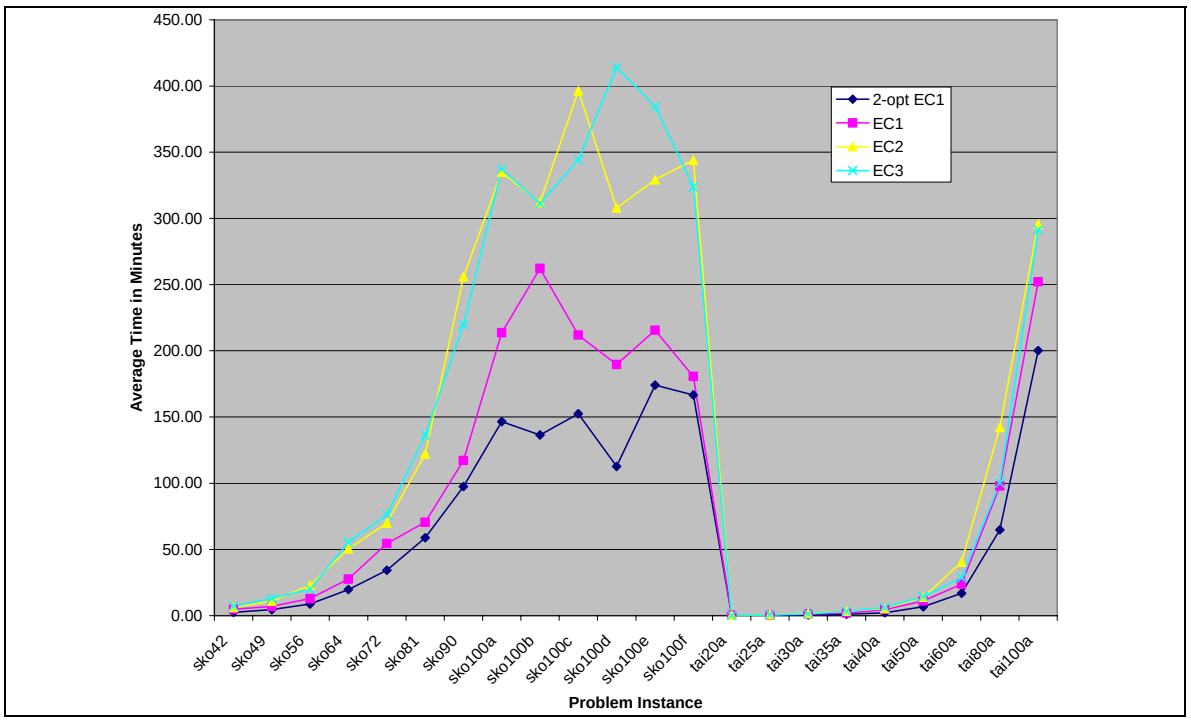


Figure 8–Average Time to Completion (ATTC) for Skorin-Kapov and symmetric Taillard instances

		2-Opt EC1				EC1				EC2				EC3			
Problem	BKS	APD	BPD	ABI	ATTS	APD	BPD	ABI	ATTS	APD	BPD	ABI	ATTS	APD	BPD	ABI	ATTS
Skorin-Kapov Instances																	
sko42	15812	0.008	0.000	753938	6.61	**0.000**	0.000	103467	1.62	0.019	0.000	1183886	27.50	**0.000**	0.000	984019	21.43
sko49	23386	0.053	0.000	1888633	26.81	0.063	0.000	615169	15.08	0.107	0.051	857422	28.00	**0.039**	0.000	15511213	50.91
sko56	34458	0.183	0.000	1651179	37.48	0.324	0.186	1127980	43.41	0.305	0.012	771265	38.07	**0.027**	0.012	1176507	58.02
sko64	48498	0.475	0.091	1106456	51.18	0.397	0.000	504155	38.14	0.139	0.008	469825	44.27	**0.078**	0.000	642791	60.99
sko72	66256	0.827	0.211	473306	30.55	0.705	0.196	593491	60.92	0.319	0.060	316919	39.52	**0.250**	0.115	635224	79.22
sko81	90998	0.855	0.086	473255	50.74	0.943	0.481	247691	41.69	0.364	0.097	327019	65.44	**0.278**	0.114	340529	68.06
sko90	115534	0.691	0.230	254279	40.27	0.785	0.533	212166	51.87	**0.424**	0.029	272543	77.96	0.473	0.132	202259	57.84
sko100a	152002	0.739	0.378	144780	33.47	0.617	0.163	110996	39.94	0.424	0.112	208584	86.13	**0.340**	0.197	171948	70.86
sko100b	153890	0.868	0.659	176445	40.82	0.521	0.396	92304	33.19	0.436	0.149	238888	98.71	**0.408**	0.140	129912	53.54
sko100c	147862	1.122	0.557	167928	38.82	1.445	0.889	132206	42.48	0.963	0.127	182385	75.53	**0.543**	0.172	188109	77.63
sko100d	149576	1.007	0.389	59199	13.68	1.019	0.784	110292	39.52	0.537	0.119	202564	83.66	**0.517**	0.182	200304	88.14
sko100e	149150	1.191	0.801	202636	46.93	0.796	0.413	101984	36.66	0.692	0.020	216867	89.67	**0.460**	0.020	214302	88.33
sko100f	149036	1.008	0.782	237391	52.01	1.063	0.625	100648	36.20	0.627	0.279	187227	77.53	**0.542**	0.421	203808	84.19
Average		0.694	0.322	583802	36.10	0.668	0.359	311734	36.98	0.412	0.082	418107	64.00	**0.304**	0.116	510840	66.09
Symmetric Taillard Instances																	
tai20a	122455319	**0.000**	0.000	18885284	7.52	**0.000**	0.000	5359412	5.80	**0.000**	0.000	410389	0.89	**0.000**	0.000	260152	0.56
tai25a	344355646	0.102	0.000	24862224	26.19	**0.000**	0.000	3924530	9.26	**0.000**	0.000	265401	1.10	**0.000**	0.000	132537	0.55
tai30a	637117113	0.402	0.016	10567411	22.58	0.161	0.000	6367375	27.75	**0.000**	0.000	520451	3.62	**0.000**	0.000	293665	2.04
tai35a	283315445	0.479	0.082	4647443	17.68	0.366	0.067	3902298	28.46	**0.000**	0.000	507435	5.51	**0.000**	0.000	915365	9.95
tai40a	637250948	0.519	0.074	5101524	31.42	0.550	0.074	2545141	28.88	0.274	0.074	1728563	27.76	**0.219**	0.074	1742277	28.01
tai50a	458821517	0.802	0.635	2988029	44.29	0.753	0.534	2209486	56.93	0.550	0.352	1045558	35.57	**0.514**	0.364	1041698	35.43
tai60a	608215054	0.883	0.769	1596279	54.50	0.791	0.403	800025	45.52	**0.629**	0.499	685710	49.25	0.657	0.336	685735	49.24
tai80a	818415043	0.745	0.559	535781	54.01	0.714	0.532	417335	65.71	**0.681**	0.554	421467	79.46	0.730	0.486	345948	65.05
tai100a	1185996137	0.567	0.354	292228	67.46	**0.558**	0.313	160611	57.79	0.714	0.630	180522	74.65	0.729	0.419	131851	54.36
Average		0.500	0.277	7719578	36.18	0.433	0.214	2854024	36.23	**0.316**	0.234	640611	30.87	0.317	0.187	616581	27.24
Overall		0.615	0.303	3502983	36.14	0.571	0.300	1351762	36.67	0.373	0.144	509131	50.45	**0.309**	0.145	554098	50.20

Table 3–Computational results for Skorin-Kapov problems and symmetric Taillard problems using Stopping Criterion 2

Table 2 shows that the ejection chain neighborhood improved the average solution quality of all but 4 problems over the 2-exchange neighborhood embedded in the same heuristic. The impact of the ejection chain neighborhood can be easily observed as EC1 obtained better average results than the 2-exchange EC1 algorithm on 19 of the 22 test problems and tied on 1. The 2-exchange EC1 and EC1 are identical algorithms except for the neighborhood utilized. In the 2-exchange EC1 algorithm, the maximum length of the chain was limited to 2 nodes, which simulates a 2-exchange neighborhood. The best overall solution was found by EC1 for 12 out of the 22 problems, with the 2-exchange version obtaining the best overall solution for 9 out of 22, with a tie for one problem where both variants found the BKS.

In Table 3 the results were similar. EC1 using the ejection chain neighborhood obtained better average results on 13 of the 22 problem instances and tied on 1. The performance was degraded a small amount due to the time limit imposed. It should be noted that the 2-exchange neighborhood algorithm is able to perform around twice as many iterations in the same amount of time as the ejection chain neighborhood. While the ejection chain neighborhood is quick compared to a full *k*-opt exploration, it is still slower than a swap neighborhood. This leads to an interesting observation. In Table 3, where the runtime of the algorithm was restricted, the ejection chain neighborhood still outperformed the 2-exchange neighborhood in terms of solution quality. On 8 of the 14 test instances where the ejection chain neighborhood tied or bested the 2-exchange neighborhood in Table 3, the average time to the best solution (ATTS) for EC1 was actually less than the 2-exchange neighborhood EC1. This indicates that the ejection chain neighborhood is able to quickly find high quality solutions. This is reinforced by the results in Table 2, which shows that allowed to iterate with stagnation as the stopping condition, EC1 using the ejection chain neighborhood performs even better against the 2-exchange neighborhood EC1. These results are obtained in most cases without doubling the computational time of the 2-exchange neighborhood EC1.

In Table 2, as well as in Table 3, the EC3 multi-start variant produced the best overall results of all approaches, obtaining the best average solution quality for 16 out of 22 problems in Table 2. In Table 3, EC3 obtained the best average solution quality for 14 out of 22 problems. However, EC2 and EC3 tied on 4 of the 22 problems, so the results indicate that EC3 does as well as or better than all other variants under SC2 on 18 of the 22 problems. EC2 had 4 of the best average percent deviations in Table 2 (3 out of 22 in Table 3). EC1 and the 2-exchange EC1 provided one best average percent deviation each in Table 2. In Table 3, the 2-exchange EC1 obtained the best average percent deviation to one problem. The EC2 variant, which replaced the current working solution with the global best solution rather than a diversified solution, was clearly outperformed by the EC3 variant. This suggests that the use of strategic diversification is highly beneficial and agrees with previous findings where metaheuristics applied to the QAP that employ some type of diversity have provided good results (Misevicius 2003, 2005; Drezner, 2003).

EC2 shows a slight edge over EC3 in obtaining the best overall solution. EC2 produced the best overall solutions (BPD) for 8 of 22 problems in Table 2 (7 out of 22 in Table 3) while EC3 produced 5 of the best overall solutions in Table 2 (3 out of 22 in Table 3). EC1 produced 1 best overall solution in both Tables. In Table 3, 2-exchange neighborhood EC1 produced 1 best overall solution. On the other 8 instances in Table 2 (10 in Table 3) at least 2 of the variants tied.

4.1 Comparisons with very-large scale neighborhood algorithms

The VLSN algorithms introduced by Ahuja et al. (2007) employ a type of variable depth *k*-opt neighborhood and are therefore appropriate for comparison with our EC algorithms. The purpose of this comparison is to investigate the relative performance of the ejection chain algorithms in the current study and other large neighborhood algorithms. In order to clarify the analysis, it is convenient to discuss the fundamentals of VLSN algorithms and contrast them to the EC algorithms.

A full path enumeration search requires that every *k*-exchange be explored and can be prohibitively expensive even for relatively small values of *k*. This expense has severely limited attempts to use neighborhoods for the QAP more complex than the swap neighborhood (which results in $k = 2$). The VLSN algorithms and the ejection chain algorithms contribute alternative approaches for exploring larger neighborhoods.
The VLSN algorithms of Ahuja et al. (2007) introduce an improvement graph for the QAP together with several variants of a search algorithm utilizing a large neighborhood. The concept of an improvement graph was first introduced by Thompson and Orlin (1989) for partitioning problems. For the QAP, it is used to store partial costs for *k*-exchanges on a permutation. The initial cost of constructing the improvement graph is $\mathcal{O}(n^3)$; however, once created for a permutation it can be updated in $\mathcal{O}(kn^2)$ time. The improvement graph does not contain the full cost of the *k*-exchange, rather a good approximation. It is especially useful in a path enumeration scheme as it allows for relatively quick evaluation of a large number of neighbors on a single permutation with the construction of only one improvement graph. If a new permutation is introduced, the improvement graph must be reconstructed.

The ejection chain method, in contrast, exploits a selective subset of neighbors, which provides a quick investigation of a promising extended neighborhood rather than examining all exchanges at a given depth. This reduces the number of calculations necessary, thus speeding the neighborhood search process without the necessity of maintaining a cost matrix. As in the VLSN method a *k*-level ejection chain does not necessarily produce the overall best *k*-exchange move, rather a potentially good move. Ejection chain strategies are particularly amenable to being exploited within an adaptive memory TS framework, where other operators may also be applied to the permutation.

The VLSN search algorithms in Ahuja et al. (2007) explore iteratively larger neighborhoods up to a defined *k* beginning from a randomly drawn permutation. Specifically, the VLSN algorithms explore all exchanges at depth 2, then all (or a pruned subset) exchanges at depth 3, followed lastly by the 4-exchanges (in implementation, the algorithms are limited to a depth of 4). In contrast, the ejection chain method discovers the best exchange at depth 2, and then iteratively extends that 2-exchange up to a depth of *n*. The method keeps track of the level k^* of the chain where the best trial solution was found and then applies the associated *k*-exchange move to the permutation and the process is repeated. An iteration of the VLSN algorithms ends when the search on the current permutation is exhausted (a new best solution is found or no better solution is found after exploring all allowed *k*-exchanges). Then a new random permutation is drawn, a new improvement graph is constructed and the process is repeated. When comparing the two methods, the VLSN algorithms more nearly resemble a breadth-first search and the EC algorithms more nearly resemble a depth-first search, though each affords the strategic benefit of avoiding the complexity of such classical searches while nevertheless uncovering high quality solutions. Future research that combines the two approaches would be of interest.

The difference between the VLSN algorithms concerns how many *k*-exchanges are examined at a given depth. The authors propose four VLSN variants. The first is a full path enumeration where all paths of a given depth are examined. As previously mentioned this process is very costly and was ruled out by the authors as a viable algorithm. The other three variants employed "path pruning" techniques to reduce the number of paths examined at each depth. The second proposed variant, keeps only paths with a negative cost at each depth. The authors state that this variant was outperformed in testing by the other variants, so computational results for this variant were not provided. In both VLSN variants, for which results are presented, a different path pruning technique is employed to reduce the number of paths examined at each depth. We will refer to these two variants as VLSN1 and VLSN2. In VLSN1, the best αn^2 paths with the lowest cost at one level are carried forward to the next level to build larger exchanges. In implementation, α is set to 1 and the maximum path length is set to 4. By contrast, VLSN2 excludes all paths except for the best path for each node. In other words, only the initial path from each node that has the lowest cost is allowed to proceed to the next depth. VLSN2 also first performed a descent to a local optimum at depth 2 before beginning path enumeration.

Table 4 provides comparisons between the two VLSN algorithms (VLSN1 and VLSN2) and the EC algorithms (EC1, EC2, and EC3) developed for this study using the same stopping criterion (SC2). The results for SC1 are also provided for some supplemental observations. In addition, we provide in Figure 9 a pairwise comparison of the algorithms in terms of the number of best solutions produced over all problems. The VLSN algorithms were run using SC2 on a Pentium IV, 2.4 GHz processor and were also written in the C programming language. SPEC (2000) shows that the processor used in the VLSN study is equivalent to (just slightly better than) the processor used in the current study (Intel Itanium, 1.3 GHz). Therefore, the comparisons are as valid as possible without having algorithms written by the same programmer and run on the same machine.

Problem	BKS	Stopping Criterion 1 (SC1)			Stopping Criterion 2 (SC2)						
		EC1	EC2	EC3	2-Opt EC1	2-Opt VLSN	EC1	EC2	EC3	VLSN1	VLSN2
		APD	APD	APD	APD	APD	APD	APD	APD	APD	APD
sko42	15812	0.028	0.293	0.061	0.008	**0.000**	**0.000**	0.019	**0.000**	**0.000**	**0.000**
sko49	23386	0.199	0.235	0.086	0.053	0.188	0.063	0.107	**0.039**	0.103	0.214
sko56	34458	0.527	0.475	0.259	0.183	0.348	0.324	0.305	**0.027**	0.116	0.226
sko64	48498	0.464	0.243	0.139	0.475	0.334	0.397	0.139	**0.078**	0.177	0.433
sko72	66256	0.691	0.322	0.340	0.827	0.426	0.705	0.319	**0.250**	0.260	0.465
sko81	90998	0.849	0.336	0.271	0.855	0.433	0.943	0.364	**0.278**	0.308	0.516
sko90	115534	0.881	0.305	0.272	0.691	0.573	0.785	0.424	0.473	**0.407**	0.457
sko100a	152002	0.661	0.314	0.263	0.739	0.524	0.617	0.424	0.340	**0.289**	0.462
sko100b	153890	0.803	0.379	0.226	0.868	0.502	0.521	0.436	0.408	**0.395**	0.550
sko100c	147862	0.779	0.731	0.269	1.122	0.498	1.445	0.963	0.543	**0.331**	0.594
sko100d	149576	0.862	0.437	0.316	1.007	0.580	1.019	0.537	0.517	**0.439**	0.619
sko100e	149150	1.147	0.522	0.198	1.191	0.654	0.796	0.692	0.460	**0.257**	0.654
sko100f	149036	0.979	0.567	0.395	1.008	0.621	1.063	0.627	0.542	**0.326**	0.652
Average		0.628	0.397	0.238	0.694	0.437	0.668	0.412	0.304	0.262	0.449
tai20a	122455319	0.718	0.152	0.199	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
tai25a	344355646	0.670	0.294	0.055	0.102	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
tai30a	637117113	0.645	0.178	0.137	0.402	0.016	0.161	**0.000**	**0.000**	**0.000**	0.177
tai35a	283315445	0.972	0.302	0.272	0.479	0.384	0.366	**0.000**	**0.000**	**0.000**	0.384
tai40a	637250948	0.825	0.420	0.387	0.519	1.160	0.550	0.274	**0.219**	0.687	1.099
tai50a	458821517	1.058	0.732	0.726	0.802	1.813	0.753	0.550	**0.514**	1.151	1.665
tai60a	608215054	1.017	0.715	0.861	0.883	2.016	0.791	**0.629**	0.657	1.400	1.746
tai80a	818415043	0.695	0.644	0.780	0.745	2.166	0.714	**0.681**	0.730	1.459	1.957
tai100a	1185996137	0.563	0.600	0.654	0.567	2.266	**0.558**	0.714	0.729	1.569	1.900
Average		0.796	0.449	0.452	0.500	1.091	0.433	0.316	0.317	0.696	0.992
Overall		0.729	0.418	0.326	0.615	0.705	0.571	0.373	0.309	0.440	0.671

Stopping criteria 1: no improving move found in 5000**n* iterations; **Stopping Criterion 2**: 1 hour for $n \le 40$, 2 hours for $n > 40$

Table 4–EC comparisons with VLSN

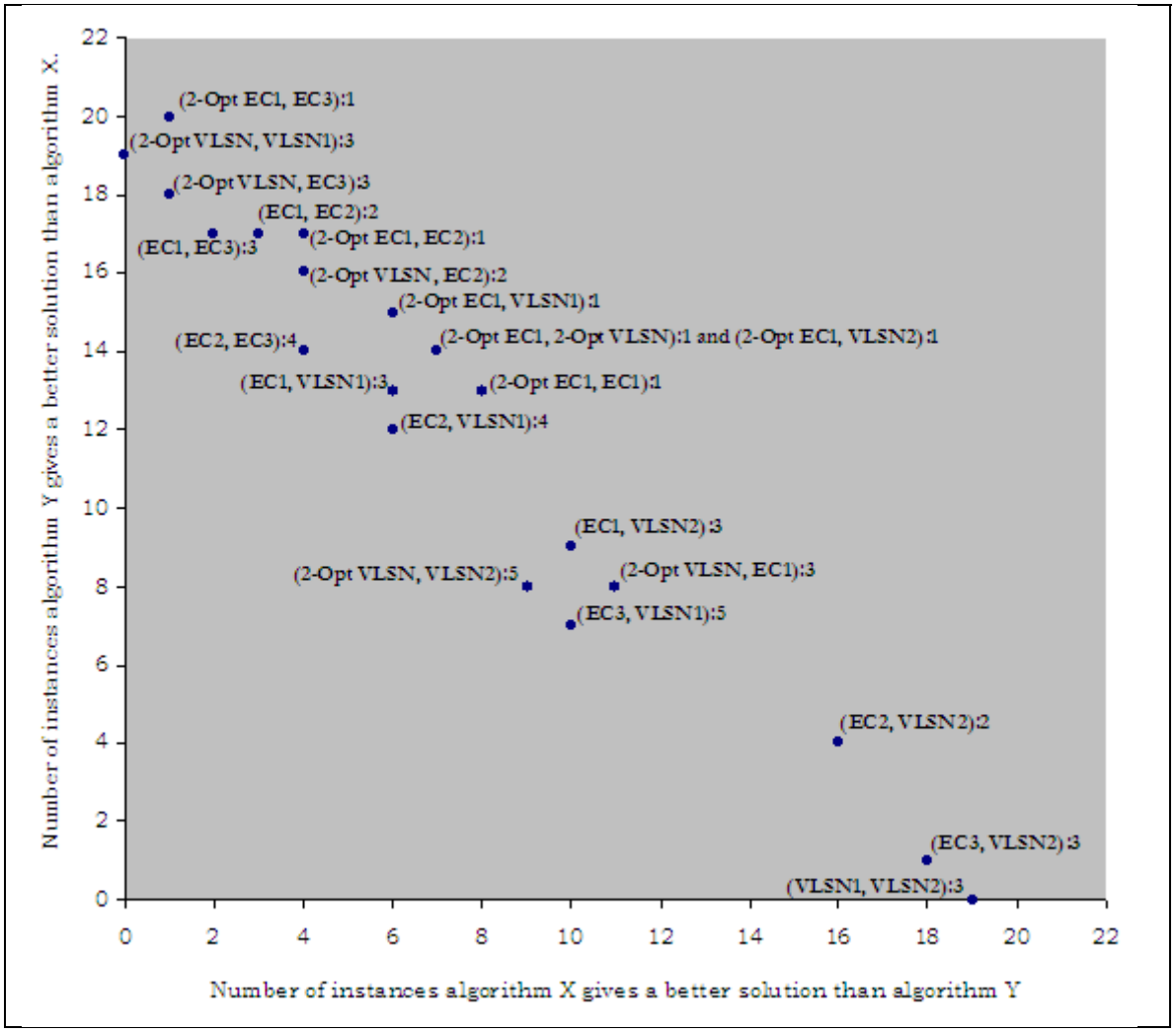


Figure 9–Number of instances one algorithm is better than another
(algorithm X, algorithm Y):number of ties

As previously discussed, the EC1 algorithms contained very limited adaptive memory guidance in the form of a simple tabu list with small tabu tenures. A 2-exchange version of EC1 is provided to examine the impact of the ejection chain neighborhood. However, since the VLSN study also provided a 2-exchange algorithm, we can compare the impact of the minimal short-term memory guidance in the EC1 variant. Comparing the 2-exchange versions of VLSN and EC1, the VLSN algorithm performs better on the sko* instances while EC1 is better on the tai* instances. Overall, there is no significant difference in the performance of the two 2-exchange algorithms, which demonstrates that the tabu list in the EC1 version had little positive impact other than to prevent cycling.

Another interesting observation can be made examining the 2-exchange algorithms. The 2-exchange EC1 algorithm did especially poorly on the sko* instances, losing on 11 of the 13 instances. This could indicate that the restrictiveness of the tabu list was especially detrimental for this set of instances. Indeed, experimental analysis

conducted during the tuning of the algorithm parameters showed better results (on average) when smaller tabu tenures are used for the sko* instances. However, those parameters were not so suitable for tai* instances and we restricted our method to use the same parameter values for all instances. In contrast, the 2-exchange VLSN revealed significant difficulty on the larger tai* instances ($n \geq 40$), exceeding the best known solutions value by more than 2%. In fact, the difference in the characteristics of the problem instances becomes especially apparent in the larger neighborhoods. This is also seen to be true when examining the difference between the variable-depth versions of VLSN and the EC algorithms.

VLSN1 performs very well on the sko* instances. It performs better than EC1 and EC2 on all but one or two instances and completely dominates VLSN2 on the sko* test set. In general the VLSN algorithms tend to perform better on the sko* instances; however even in that set both EC2 and EC3 perform better than VLSN2. Only EC3 is competitive with VLSN1, obtaining better solutions on 5 of the 13 sko* instances and tying on 1. VLSN1 obtains the best quality solutions on the biggest 7 sko* instances under SC2.

VLSN1 introduces diversity into the search by carrying forth the best n^2 paths regardless of their cost benefit. EC3 brings diversity into the search by introducing variability in tabu tenure ranges and allowing for multi-starts. This indicates that the interplay between intensification and diversification of the search may be especially important for this set of instances. On one hand, restricting the local search may prevent the methods from reaching good local optimal solutions that are relatively close in the solution space, but not necessarily within the neighborhood space of the current 2-exchange neighborhood. On the other hand, some appropriate level of diversification should be maintained in order for the method to explore other regions of the solution space.

The positive results of VLSN1 for the sko* set seem to indicate that for these instances a strategy that simultaneously explores intensification and diversification may be more appropriate than strategies that alternate between the two search strategies. This speculation is also supported by experiments on landscape analysis for the QAP under a 2-exchange neighborhood (e.g. Merz and Freisleben, 2000; Stützle, 2006). These studies show that sko* instances have a smooth landscape with a significantly high correlation between neighboring solutions. However, local optima distributions show that good solutions are spread out across the solution space. Hence, the challenge in these instances is not so much in escaping from local optimality but rather in determining the regions where the best local optima actually exist. Since local optima in these landscapes share some degree of similarity it is unlikely that good local optima exist in the vicinity of a relatively poor local optimum. Although this may establish a sufficient condition for an algorithm to engage in a stronger diversification search, a region of high-quality solutions is not necessarily close to a best local optimum. In any case, the best local optima are likely to be encountered in regions of relatively high-quality solutions, thus suggesting a somewhat extensive exploration of the search around these potentially good regions. We conjecture that the relative advantage of VLSN1 on these sko* instances stem from its breath-first type of local search strategy. In a depth-first search strategy as in our EC algorithms these issues are addressed by giving the algorithm sufficient flexibility to explore multiple search paths from local regions, thus suggesting the advisability of a small tabu tenure as considered in our EC1, perhaps just large enough to prevent cycling. In this algorithm intensification is promoted by using small tabu tenure ranges and diversification is achieved by the long search paths generated by the ejection chain neighborhood. The interplay between intensification and diversification is obtained by the variable depth moves selected by the ejection chain algorithm. Short moves keep the search in the vicinity of the current

region while long moves induce the search to explore other regions. A more aggressive interaction between intensification and diversification results from coupling the inherent depth-first search of the EC neighborhood with the desirable breath-first component emphasized in VLSN. This is accomplished in EC2 and EC3 algorithms by combining larger tabu tenures with an aspiration criterion, and in addition allowing for multi-starts. On one hand, large tabu tenures implement stronger diversification. On the other hand, the aspiration prevents the algorithm from overlooking best solutions while keeping the balance between intensification and diversification.

The symmetric tai* instances appear to behave differently. Both EC2 and EC3 beat or tie both VLSN algorithms on all 9 problems. EC1 outperforms VLSN1 on 5 of the 9 instances, ties on 2, and outperforms or ties VLSN2 on all 9 instances. For this set of instances, the interplay between intensification and diversification does not prove to be as influential. This may be justified by the fact that tai* instances have a highly rugged fitness landscape structure with far more local optima than sko* instances. Although in both sko* and tai* test sets local optimal distributions show good solutions spread out all over the solution space, the (almost) nonexistent correlation between neighboring solutions in tai* instances makes them more difficult than sko* instances when local searches are limited to 2-exchange neighborhoods. Since local optima in the tai* instances are typically very deep and share no similarities, extending the depth in k-exchange neighborhoods (to high values of k) may be more beneficial than limiting it (to small values of k) in order to make it possible to explore each level (k) of the neighborhood more extensively. We conjecture that this is what gives the edge to the EC algorithms over their VLSN counterpart on the tai* instances. In fact, the best overall solutions for the larger symmetric tai* instances are split between the EC algorithms. This trend was exhibited in computational tests where parameter settings or adaptive memory guidance could be modified to improve results on one test set to the detriment of the other. With the simple adaptive memory guidance employed in this study, the parameter settings used were found to provide the best compromise in solution quality between the two test sets. Future work could include using more sophisticated adaptive memory techniques to overcome this characteristic.

The results presented in Tables 2, 3, and 4 demonstrate the impact of the ejection chain neighborhood structure. The significance of the larger embedded neighborhoods is demonstrated by the improvement of the results obtained by EC1 over the same algorithm limited to a 2-exchange neighborhood. EC1 implemented a very simple local search with short tabu tenure and no restriction on depths (levels) explored. In an overall analysis, the EC algorithms are very competitive with the VLSN algorithms. EC1 performs better than VLSN2 in terms of both solution quality and number of best solutions, but it is not as good as VLSN1. EC2 provides better average solution quality than VLSN1, though VLSN1 manages to find a greater number of best solutions. EC2 and EC1 provide better results on 6 of the 22 instances where VLSN1 provides better results on 12 of the 22 instances against EC2 and 13 of the 22 instances against EC1, tying on the others. EC3 provides better results on 10 of the 22 instances against VLSN1. EC3 and VLSN1 tie on 5 instances and VLSN1 provides better results on 7 instances.

VLSN2 does not perform as well as the EC algorithms and is not competitive with VLSN1. Comparing VLSN2 to the EC algorithms, the worst EC variant (EC1) obtains better results to 10 out of 22 instances. The two algorithms tie on 3 instances and VLSN2 wins on 9. EC2 obtains better quality results than VLSN2 on 16 of the 22 instances and ties on 2. VLSN2 obtains the best result on the remaining 4 instances against EC2. EC3 wins on 18 of the 22 problem instances, they tie on 3, and VLSN2 obtains the best result on one instance.

In summary, no algorithm dominates all the others on all problem instances, and EC and VLSN seem to be more effective on different test sets. However, EC3 is the overall winner in terms of average solution quality and number of best solutions. These results suggest there may be significant value in combining the EC and VLSN strategies, and that additional adaptive memory guidance can be useful for further improving the EC approaches.

The results for the EC algorithms using SC1 are also shown in Table 4. This provides the opportunity to view the difference in solution quality between stopping conditions in the EC algorithm, disclosing that with longer runtimes significant improvement in solution quality can be obtained.

4.2 Extended computational analysis

We now extend our analysis to include comparisons with traditional Iterative Local Search (ILS) and Ant Colony Optimization (ACO) algorithms, which have some similarities to our multi-start TS algorithms. Comparisons to several of the best of the more complex metaheuristic algorithms are also given. Tables 5 and 6 provide results for the following additional algorithms from the literature:

- Robust Tabu Search – RTS (Taillard, 1991)
- Four Iterated Local Search Variants – ILS1, ILS2, ILS3, ILS4 (Stützle, 2006)
- Three Ant Colony Optimization Variants – ACO1, ACO2, ACO3 (Stützle and Dorigo 1999)
- A Genetic Algorithm Hybrid with a modified RTS (GA/MRT) (Drezner 2008)
- An Ant Colony Optimization/Genetic Algorithm/Local Search Hybrid – ACO/GA/LS Tseng, and S. Liang (2003)
- Three Tabu Search variants –ETS1, ETS2, and ETS3 (Misevicius, 2005)
- Two Population Based ILS Algorithms – ILS5 and ILS6 (Stützle, 2006)
- An Improved Population Based ILS Algorithm – I-ILS6 (Stützle, 2006)

These algorithms were all run on different platforms utilizing different stopping conditions. Therefore, time comparisons cannot be provided. The best ejection chain algorithm (EC3) and the best VLSN algorithm (VLSN1) are also shown in the tables both using stopping criterion SC2 for consistency.

Table 5 provides a comparison of the three EC algorithms developed for this study with the results obtained for the classical robust tabu search (RTS) algorithm (Taillard, 1991), four variants of the iterative local search (ILS) algorithm (Stützle, 2006), and three variants an ant colony optimization (ACO) algorithm (Stützle and Dorigo, 1999). These algorithms are most comparable to EC3 and VLSN2 in terms of structure and the heuristic guidance employed. Not all algorithms provide solutions for all test instances, so comparisons are only shown for the overlapping instances. Dash symbols indicate that results were not provided for that instance by the corresponding algorithm. The average solution quality over all the instances tested by the corresponding algorithm is provided at the bottom of each test set. Similar averages over all problems tested are provided at the bottom of the table.

Although limited to short-term memory components of tabu search and 2-exchange neighborhoods, RTS has long been one of the most successful tabu search algorithms for the QAP. Perhaps, due to its excellent tradeoff between algorithmic simplicity and

solution quality, RTS is often used in a large variety of more complex algorithms such as those discussed later.

All ILS variants use 2-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 (ILS1), 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. In the second version (ILS2), a random restart is employed which straightforwardly replaces the working solution with a random permutation. The third variant (ILS3) always perturbs the working solution obtained from the local search. The fourth variant (ILS4) allows worse solutions based upon a probability, that are then perturbed and the local search restarted. Several population-based variants of ILS (ILS5, ILS6, I-ILS6) are also proposed. 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.

ACO uses probabilistic perturbations that build solutions by choosing an assignment influenced by the search history (pheromone trail). A local search is then applied to the constructed solution. The first variant (ACO1) modifies the construction phase to use the pheromone trail to modify the current solution rather than construct a new one. The other two variants use a typical ACO construction phase but ACO2 applies a 2-opt local search and ACO3 uses RTS as its local search. The type of memory used in these ACO algorithms is obviously more complicated than that used by the previous algorithms, including RTS.

As we can see in Table 5, EC3 is very competitive with RTS. Both algorithms use simple short-term memory based on tabu tenure restrictions and two levels of aspiration. RTS beats EC3 in all sko* instances and ties in the only solution where both algorithms manage to find the best known solution. The reverse situation occurs for the tai* instances where EC3 ties RTS on the three instances where RTS finds the best known solution and completely dominates RTS on the remaining instances, yet finding one more best known solution. The fact that VLSN1 is not as good as RTS on either of the test sets reinforces the idea that the depth of the neighborhood is particularly relevant. When comparing EC3 to the ILS and ACO algorithms in Table 5, EC3 appears very competitive on the sko* instances and completely dominates all four ILS variants and the three ACO variants on the tai* instances. These results strongly uphold our conjecture on the potential advantage of the ejection chain neighborhood over the 2-exchange neighborhood for the tai* instances. For algorithms using the same neighborhood structure, the starting solution seems to have an effect on the quality of the solutions produced. The best ILS variant (ILS2) using random restart is superior to the other three variants that always perturb some solution previously found during the search. Since a small perturbation is always applied to the local optimum found in an iteration of the ILS algorithm is seems quite natural that a stronger diversification may be needed at a restart. This requirement does not seem so relevant when larger neighborhoods are used. For example, ILS1 and EC3 both restart the search by perturbing the current best solution; however, EC3 significantly outperforms ILS1 on all instances of both test sets. Also, both EC3 and VLSN1 outperform ILS2 on average over all problems tested.

		EC3	VLSN1	RTS	ILS1	ILS2	ILS3	ILS4	ACO1	ACO2	ACO3
Problem	BKS	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD
sko42	15812	0.000	0.000	0.000	0.269	0.010	0.010	0.161	0.076	0.015	0.104
sko49	23386	0.039	0.103	0.038	0.226	0.133	0.133	0.139	0.141	0.067	0.150
sko56	34458	0.027	0.116	0.010	0.418	0.087	0.087	0.153	0.101	0.068	0.118
sko64	48498	0.078	0.177	0.005	0.413	0.068	0.068	0.202	0.129	0.042	0.171
sko72	66256	0.250	0.260	0.043	0.383	0.134	0.134	0.294	0.277	0.109	0.243
sko81	90998	0.278	0.308	0.051	0.586	0.101	0.100	0.194	0.144	0.071	0.223
sko90	115534	0.473	0.407	0.062	0.576	0.131	0.187	0.322	0.231	0.192	0.288
sko100a	152002	0.340	0.289	0.089	0.358	0.115	0.161	0.257	-	-	-
sko100b	153890	0.408	0.395	0.056	-	-	-	-	-	-	-
sko100c	147862	0.543	0.331	0.031	-	-	-	-	-	-	-
sko100d	149576	0.517	0.439	0.055	-	-	-	-	-	-	-
sko100e	149150	0.460	0.257	0.041	-	-	-	-	-	-	-
sko100f	149036	0.542	0.326	0.066	-	-	-	-	-	-	-
Average		0.304	0.262	0.042	0.404	0.097	0.110	0.215	0.157	0.081	0.185
EC3		0.304	0.304	0.304	0.186	0.186	0.186	0.186	0.164	0.164	0.164
VLSN1		0.262	0.262	0.262	0.208	0.208	0.208	0.208	0.196	0.196	0.196
tai20a	122455319	0.000	0.000	0.000	0.723	0.503	0.542	0.467	0.675	0.191	0.428
tai25a	344355646	0.000	0.000	0.000	1.181	0.876	0.896	0.823	1.189	0.488	1.751
tai30a	637117113	0.000	0.000	0.000	1.304	0.808	0.989	1.141	1.311	0.359	1.286
tai35a	283315445	0.000	0.000	0.112	1.731	1.110	1.113	1.371	1.762	0.773	1.586
tai40a	637250948	0.219	0.687	0.462	2.036	1.319	1.490	1.491	1.989	0.933	1.131
tai50a	458821517	0.514	1.151	0.882	2.127	1.496	1.491	1.968	2.800	1.236	1.900
tai60a	608215054	0.657	1.400	0.974	2.200	1.498	1.692	2.081	3.070	1.372	2.484
tai80a	818415043	0.730	1.459	1.065	1.775	1.198	1.200	1.576	2.689	1.134	2.103
tai100a	1185996137	0.729	1.569	1.071	-	-	-	-	-	-	-
Average		0.317	0.696	0.507	1.635	1.101	1.177	1.365	1.936	0.811	1.584
EC3		0.317	0.317	0.317	0.265	0.265	0.265	0.265	0.265	0.265	0.265
VLSN1		0.696	0.696	0.696	0.587	0.587	0.587	0.587	0.587	0.587	0.587
Overall		0.309	0.440	0.232	1.019	0.599	0.643	0.790	1.106	0.470	0.931
EC3		0.309	0.309	0.309	0.225	0.225	0.225	0.225	0.218	0.218	0.218
VLSN1		0.440	0.440	0.440	0.397	0.397	0.397	0.397	0.405	0.405	0.405

Table 5–Comparisons with multi-start algorithms

Table 6 presents results for some of the best performing tabu search algorithms from the literature as well as the best performing hybrid genetic algorithms and population-based iterative local search algorithms. This set of algorithms, including the ETS implementations, ACO/GA/LS, the population-based ILS algorithms, and GA/MRT, are some of the more sophisticated and complex heuristics for the QAP. Table 6 uses the same format as Table 5. The ETS algorithms obtain some of the best results for the symmetric tai* instances and the hybrid GAs due to Drezner obtain some of the best results for the sko* instances. Population-based ILS and ACO/GA/LS hybrid algorithms perform very well on sko* instances competing closely with several GA and TS hybrid algorithms from the literature, but not as well as GA/MRT. Also, they are not as competitive on the tai* instances. The information in Table 6 is only intended to provide a cursory overview of the solution quality of the extended neighborhood algorithms in contrast to some of the best performing algorithms from the literature.

ETS1, ETS2, and ETS3 are all modified RTS algorithms embedded in multi-start tabu search approaches using a variety of diversification operators. These tabu search algorithms modify Taillard's RTS by removing the aspiration criteria, decreasing the tabu tenure, and simplifying the tabu conditions. 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. The algorithms differ by the type and combinations of the perturbation operators applied during the search. The layering in these TSs are more complicated than those in the current study, as often several levels of restarting occur with multiple diversification operators. The ETS algorithms are currently the best performing algorithms for the symmetric tai* instances but report no results for the sko* instances. EC3, even though much simpler, approximates quite well the solution quality achieved by the ETS algorithms.

The genetic algorithms due to Drezner are the most successful algorithms for the sko* test instances. Drezner has presented a series of hybrid GAs for the QAP. The algorithms differ by the improvement operator used to hybridize the GA. Drezner (2002) presents three hybrid GAs, the first uses only a strict decent operator to improve the solutions created by the GA. In the second hybrid GA, the strict decent operator is replaced with a simple tabu search. In the third hybrid GA, the incorporation of a new tabu search algorithm, concentric tabu search (Drezner, 2002), proved very successful on the sko* problems. Concentric tabu search was improved in a subsequent study (Drezner, 2003), to allow more moves than the original version and again embedded it within a GA.

Concentric tabu search shares some commonalities with the path-relinking concept. In concentric tabu search, series of swap moves are iteratively applied to a permutation until the distance of the working solution is maximally different from the original solution (or an improved solution is found). In a sense, the "center" solution is serving as both the solution initiating the search (the initiating solution in path-relinking terminology) and the solution being modified. The "path" the solutions are following is guided by the requirement that the solution be different from the original solution. A move can contribute a point (or two) to the distance (difference) score if the exchange moves at least one (or two) facilities to locations they did not previously occupy. Since in concentric tabu search, the reference set is open to all neighboring solutions that increase the difference from the "center" solution rather than restricted to a pre-selected subset of reference (or guiding) solutions, a larger number of intermediate solutions are available than in traditional path-relinking.

The algorithm works by examining all swaps on the "center" solution thereby obtaining all permutations that are 2 elements different than the "center" solution. A pruning technique, like those applied in VLSN, is used to reduce the number of permutations of distance 2 that are carried forward. The pruning technique keeps a defined number *K* of the best permutations of distance 2. Swap moves are then performed again on the *K* solutions retained. Performing a swap on the permutations of distance 2 may result in a permutation with distance 3 or distance 4. As new solutions are created, a defined number of the best solutions at a given distance are kept in lists. Once the search of all permutations at a given distance is completed, the next distance is explored, and so on. The algorithm continues this process until either a new best solution is found which restarts the process, or the maximum distance from the original "center" solution is reached. All moves made in this algorithm are swaps and the cost calculations described in Taillard (1991) and Burkard and Rendl (1984) are therefore used. This approach is novel for a QAP tabu search in that the moves made are guided by the distance from the original permutation.

While the concentric tabu search has proven to be a successful addition to a hybridized GA, for the solution of the sko* test instances, Drezner (2008) provided even better solutions to this test set using a very slightly modified RTS (MRT) incorporated into a GA (GA/MRT). The only change made to the RTS in GA/MRT is to increase the tabu tenure range. The QAP appears to be very sensitive to the parameters and adaptive memory guidance utilized both in the algorithms developed in the current study and those from the literature. Since GA/MRT provides the best results of the series, the earlier Drezner hybrid GAs are not included in the tables below. GA/MRT provides superior results to all other algorithms on the sko* instances. Results are not provided for the hybrid GAs on the tai* instances.

When comparing the large neighborhood algorithms to the population-based ILS and ACO/GA/LS hybrids, we can see that these algorithms perform better than EC3 and VLSN1 on the sko* test set, but they all lose against EC3 on the tai* instances. EC3 outperforms ACO/GS/LS on all 9 tai* instances. EC3 also outperforms ILS5 and ILS6 in all 9 tai* instances and wins on all but the 2 largest instances against I-ILS6. VLSN1 wins against ILS5 and ILS6 on the tai* instances, but it is not competitive with ACO/GS/LS or I-ILS6.

These results suggest promise for the exploration of extended neighborhoods.

		EC3	VLSN1	ETS1	ETS2	ETS3	ILS5	ILS6	I-ILS6	ACO/GA/LS	GA/MRT
Problem	BKS	APD	APD	APD	APD	APD	APD	APD	APD	APD	APD
sko42	15812	0.000	0.000	-	-	-	0.022	0.000	0.000	0.000	0.000
sko49	23386	0.039	0.103	-	-	-	0.090	0.068	0.000	0.060	0.000
sko56	34458	0.027	0.116	-	-	-	0.102	0.071	0.000	0.010	0.000
sko64	48498	0.078	0.177	-	-	-	0.079	0.057	0.000	0.000	0.000
sko72	66256	0.250	0.260	-	-	-	0.139	0.085	0.000	0.020	0.000
sko81	90998	0.278	0.308	-	-	-	0.100	0.082	0.001	0.030	0.000
sko90	115534	0.473	0.407	-	-	-	0.262	0.128	0.007	0.040	0.000
sko100a	152002	0.340	0.289	-	-	-	0.191	0.109	0.006	0.020	0.000
sko100b	153890	0.408	0.395	-	-	-	-	-	0.012	0.010	0.000
sko100c	147862	0.543	0.331	-	-	-	-	-	0.007	0.000	0.000
sko100d	149576	0.517	0.439	-	-	-	-	-	0.002	0.030	0.000
sko100e	149150	0.460	0.257	-	-	-	-	-	0.021	0.000	0.000
sko100f	149036	0.542	0.326	-	-	-	-	-	0.037	0.030	0.001
Average		0.304	0.262				0.123	0.075	0.007	0.019	0.000
EC3		0.304	0.304				0.186	0.186	0.304	0.304	0.304
VLSN1		0.262	0.262				0.208	0.208	0.262	0.262	0.262
tai20a	122455319	0.000	0.000	0.000	0.000	0.000	0.500	0.344	-	0.110	-
tai25a	344355646	0.000	0.000	0.037	0.000	0.015	0.869	0.656	0.000	0.290	-
tai30a	637117113	0.000	0.000	0.003	0.041	0.000	0.707	0.668	0.000	0.340	-
tai35a	283315445	0.000	0.000	0.000	0.000	0.000	1.010	0.901	0.000	0.490	-
tai40a	637250948	0.219	0.687	0.167	0.130	0.173	1.305	1.082	0.280	0.590	-
tai50a	458821517	0.514	1.151	0.322	0.354	0.388	1.574	1.211	0.610	0.850	-
tai60a	608215054	0.657	1.400	0.570	0.603	0.677	1.622	1.349	0.820	0.030	-
tai80a	818415043	0.730	1.459	0.321	0.390	0.405	1.219	1.029	0.620	0.860	-
tai100a	1185996137	0.729	1.569	0.367	0.371	0.441	-	-	0.690	0.800	-
Average		0.317	0.696	0.199	0.210	0.233	1.101	0.905	0.378	0.484	
EC3		0.317	0.317	0.317	0.317	0.317	0.265	0.265	0.356	0.317	
VLSN1		0.696	0.696	0.696	0.696	0.696	0.587	0.587	0.783	0.696	
Overall		0.309	0.440				0.612	0.490	0.148	0.210	
EC3		0.309	0.309				0.225	0.225	0.324	0.309	
VLSN1		0.440	0.440				0.397	0.397	0.461	0.440	

Table 6–Comparisons with advanced metaheuristic algorithms

5. Conclusions

This study examined the use of ejection chains for the QAP. The results indicate the use of these embedded neighborhood structures result in higher solution quality than obtained by using the traditional 2-exchange neighborhood. We also demonstrate the power of coupling this neighborhood definition with more sophisticated adaptive memory guidance. Our resulting ejection chain approaches are shown to be competitive with recently proposed path enumeration techniques embodied in very large search neighborhood (VLSN) methods.

Future studies could examine integrating the ejection chain algorithms with the VLSN methods and with higher level adaptive memory techniques such as path relinking. Our computational testing showed the parameters chosen for the tabu search framework and the multi-start variants produced the best results from the several combinations examined, but more thorough analysis of these parameters could also provide better quality results, particularly by means of dynamic parameter manipulation. The addition of the diversification method in the EC3 algorithm disclosed the importance of strategic diversification to find enhanced solutions, and we anticipate that additional attention to diversification strategies may also yield gains for future algorithms.

References

Ahuja, R., Jha, K., Orlin, J., and Sharma, D., (2007) "Very Large-Scale Neighborhood Search for the Quadratic Assignment Problem," *INFORMS Journal on Computing, 19(4), 646-657*.

Ahuja, R., Orlin, J., and Tiwari, A., (2000) "A Greedy Genetic Algorithm for the Quadratic Assignment Problem," *Computers & Operations Research*, 27, 917-934.

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

Cavique, L., Rego, C., and Themido, I. (1999) "Subgraph Ejection Chains and Tabu Search for the Crew Scheduling Problem," *Journal of Operational Research Society*, 50, 608–616.

Burkard, R., Karisch, S. and Rendl, F. (1997) "QAPLIB – A Quadratic Assignment Problem Library," *Journal of Global Optimization*, 10, 391-403.

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

Cung, V-D., Mautor, T., Michelon, P., Tavares, A. (1996) "Scatter Search for the Quadratic Assignment Problem," *Proceedings of the IEEE International Conference on Evolutionary Computation*, IEEE Press, 165-169.

Drezner, Z. (2005) "The Extended Concentric Tabu for the Quadratic Assignment Problem," *European Journal of Operational Research*, 160, 416-422.

Drezner, Z. (2003) “A New Genetic Algorithm for the Quadratic Assignment Problem,” *INFORMS Journal on Computing*, 15(3), 320-330.

Drezner, Z. (2002) “A New Heuristic for the Quadratic Assignment Problem,” *Journal of Applied Mathematics and Decision Sciences*,” 6(3), 143-153.

Gamboa, D., Rego, C. and Glover, F. (2005) “Data Structures and Ejection Chains for Solving Large-Scale Traveling Salesman Problems,” *European Journal of Operational Research*, 160, 154-171.

Fleurent, C. and Glover, F. (1999) "Improved Constructive Multistart Strategies for the Quadratic Assignment Problem Using Adaptive Memory," *INFORMS Journal on Computing*, 11(2), 198-204.

Glover, F. and Laguna, M. (1997) *Tabu Search*, Kluwer Academic Publishers.

Glover, F. (1998) “A Template for Scatter Search and Path Relinking,” *Artificial Evolution*, J.-K. Hao et al., eds., LNCS 1363, Springer-Verlag, 13-54.

Glover, F. (1991) “Multilevel Tabu Search and Embedded Search Neighborhoods for the Traveling Salesman Problem,” Manuscript, Leeds School of Business, University of Colorado, Boulder, CO.

Glover, F. (1992) “New Ejection Chain and Alternating Path Methods for Traveling Salesman Problems,” *Computer Science and Operations Research*, 449-509.

Glover, F. (1996) “Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems,” *Discrete Applied Mathematics*, 65, 223-253.

James, T., Rego, C., and Glover, F. (2005) “Sequential and Parallel Path-Relinking Algorithms for the Quadratic Assignment Problem,” *IEEE Intelligent Systems*, 20(4), 58-65.

Koopmans, T., and Beckmann, M. (1957) “Assignment Problems and the Location of Economic Activities,” *Econometrica*, 25(1) 53-76.

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

Merz, P., and Freisleben, B. (2000) “Fitness landscape analysis and memetic algorithms for the quadratic assignment problem,” *IEEE Trans. Evolutionary Computation* 4(4), 337-352.

Misevicius, A. (2005) “A Tabu Search Algorithm for the Quadratic Assignment Problem,” *Computational Optimization and Applications*, 30, 95-111.

Misevicius, A. (2004) “An Improved Hybrid Genetic Algorithm: New Results for the Quadratic Assignment Problem,” *Knowledge-Based Systems*, 17, 65-73.

Misevicius, A. (2003) “Genetic Algorithm Hybridized with Ruin and Recreate Procedure: Application to the Quadratic Assignment Problem,” *Knowledge-Based Systems*, 16, 261-268.

Oliveira, C.A., Pardalos, P.M., and Resende, M.G.C. (2004) “GRASP with path-relinking for the quadratic assignment problem, "Efficient and Experimental Algorithms," C.C. Ribeiro and S.L. Martins (Eds.), *Lecture Notes in Computer Science*, vol. 3059, 356-368, Springer-Verlag.

Rego, C. (1998a) “Relaxed Tours and Path Ejections for the Traveling Salesman Problem,” *European Journal of Operational Research*, 106, 522-538.

Rego, C. (1998b) A Subpath Ejection Method for the Vehicle Routing Problem. *Management Science*, 44:10, 1447-1459.

Rego, C. (2001) “Node Ejection Chains for the Vehicle Routing Problem: Sequential and Parallel Algorithms”, *Parallel Computing*, 27, 201-222.

SPEC. (2000). “SPEC Benchmark Results”, http://www.spec.org.

Stützle, T. and Dorigo, M. (1999) “ACO Algorithms for the Quadratic Assignment Problem,” *New Ideas for Optimization*, D. Corne, M. Dorigo, and F. Glover, eds., McGraw-Hill, 33-50.

Stützle, T. (2006) “Iterative Local Search for the Quadratic Assignment Problem,” *European Journal of Operational Research*, 174, 1519-1539.

Taillard, E. (1991) “Robust Taboo Search for the Quadratic Assignment Problem,” *Parallel Computing*, 17, 443-455.

Tseng, L., and and Liang, S. (2005) “A hybrid metaheuristic for the quadratic assignment problem,” *Computational Optimization and Applications*, 34(1), 85-113.

Yagiura, M. Ibaraki, T. and Glover, F. (2004) “An Ejection Chain Approach for the Generalized Assignment Problem,” *INFORMS Journal on Computing*, 16, 2, 133-151.