Journal of Heuristics, 2: 169–179 (1996)

Finding a Best Traveling Salesman 4-Opt Move in the Same Time as a Best 2-Opt Move

FRED GLOVER
US West Chaired Professor of Systems Science, School of Business, CB 419, University of Colorado, Boulder, CO 80309-0419 *email: fred.glover@colorado.edu*

Abstract

A special class of 4-opt moves plays a key role in several leading heuristics for the traveling salesman problem (TSP). However, the number of such moves is quite large—$O(n^4)$ for a graph of n nodes, on the order of the square of the number of 2-opt moves. Consequently, classical TSP heuristics have not attempted to seek best (and often not even improving) instances of these moves. We show that a best move from the collection that consists of these moves, together with an additional class of 4-opt moves and certain related 3-opt moves, can nevertheless be found in the same order of time required to find a best 2-opt move. Our method employs an acyclic shortest path model based on ideas introduced with ejection chain procedures and generates a sequence that can include improving moves at earlier stages. Joined with candidate list strategies that limit the tour edges available to be dropped, the method can also be structured to find best members from the set of implied surviving moves in $O(n)$ time, making available TSP strategies for incorporating 4-opt moves that were previously beyond practical consideration.

1. Introduction

A variety of heuristics for the traveling salesman problem (TSP) incorporate special instances of 4-opt moves as a fundamental part of their design (see, for example, Martin, Otto, and Felten, 1992; Fiechter, 1994; Johnson and McGeoch, 1996). However, these moves are significantly more numerous than many other popularly used moves, presenting $O(n^2)$ times as many possibilities as 2-opt moves for a graph of n nodes. Consequently, most efforts to exploit these 4-opt moves do not undertake to find best or even improving instances of them. To data, it appears that no methods have sought to identify local optima relative to such moves.

We show, however, that best instances of these 4-opt moves and a related class of additional 4-opt moves (with twice as many members) can be found in the same order of time as required to find best instances of 2-opt moves. Specifically, we provide a method that finds a best move from a collection that embraces these two classes of 4-opt moves and also an associated class of 3-opt moves, together with the class of all 2-opt moves. The approach is based on an acyclic shortest path construction, following ideas introduced with ejection chain methods for TSPs (Glover, 1992). The construction also allows the possibility of finding improving moves during the process of finding a best move. Specializations for sparse graphs afford an opportunity to reduce the worst case bound. In particular, implemented in conjunction with candidate list strategies that select a bounded subset of tour edges as

candidates to be deleted, the method can be structured to find a best move from the set of implied survivors in $O(n)$ time.

2. Definitions and notation

We focus attention on the symmetric TSP, whose goal is to find a least-cost Hamiltonian cycle in an undirected graph $G = (N, E)$, where $|N| = n$. For convenience of description we will suppose G is dense—that is, that the edge set E consists of all (unordered) pairs of nodes of N. Our approach can also be applied to asymmetric TSPs by appropriately redefining evaluations, without changing the computational bounds. We assume the reader has a rudimentary acquaintance with standard graph theory terminology and dispense with formal definitions where context makes our meaning apparent.

Let T denote an arbitrary tour and assume the nodes of N are indexed so they are visited by the edges of T in the natural order from 1 to n. (Some TSP heuristics can be implemented more efficiently by special data structures that do not maintain the tour in such an oriented form. This does not affect order bounds on computation but can provide a practical enhancement.) We refer to ith edge, $(i, i + 1)$ of T as $e(i)$, for $i = 1$ to n, and adopt the convention that the index $n + 1$ corresponds to 1, hence $e(n)$ is the edge $(n, 1)$.

2.1. *Simple moves and alternating paths*

Neighborhoods for transforming one TSP tour into another are often cataloged in terms of k-opt moves, which are operations that delete k tour edges and add k non-tour edges to create a new tour. This is a nonstructural move classification, which says nothing about the relationship among the edges deleted and added.

A more informative characterization of TSP neighborhoods can be achieved by reference to alternating paths and cycles, as introduced originally by Berge (1962). Specifically, as shown in the ejection chain development of Glover (1992), the symmetric difference between any two TSP tours can be expressed as an edge disjoint collection of alternating cycles, and a special "parsimonious neighborhood" can be identified that precisely generates such cycles while maintaining an associated transformation to yield legitimate TSP tours as trial solutions at each step. (The result holds for both symmetric and asymmetric TSPs, by interpreting a cycle to be undirected or directed, as appropriate.)

We call an alternating cycle (AC) that drops k tour edges and adds k nontour edges a k-AC. (For convenience, we speak interchangeably of an AC and the transformation of T that it includes.) Such a cycle will be called *connecting* if it transforms T into a connected subgraph and *disconnecting* otherwise. We will state several observations that are easily established, and whose specific underlying transformations can be generated directly by the ejection chain framework.

- *Remark 1.* A k-AC transforms T into a new tour and hence constitutes a k-opt move, if and only if it is connecting.
- *Remark 2.* A k-opt move in general is composed of a collection of edge-disjoint h-ACs for various positive values of h that sum to k. These component h-ACs can include disconnecting members.

- *Remark 3.* A necessary and sufficient condition for a collection of edge-disjoint ACs to yield a tour and hence to define a k-opt move, is that they transform T into a connected subgraph.

The simplest alternating cycles are 2-ACs, and the connecting 2-AC is the one that goes by the popular "2-opt" designation. Relative to the specified indexing of the nodes of T, connecting and disconnecting 2-ACs can be completely determined by identifying the two dropped edges, $e(i) = (i, i+1)$ and $e(j) = (j, j+1)$ for $i = 1$ to $n-2$ and $j = i+2$ to n, as follows:

- *Connecting* 2-AC (2-opt): drop $e(i)$ and $e(j)$, add (i, j) and $(i+1, j+1)$
- *Disconnecting* 2-AC: drop $e(i)$ and $e(j)$, add $(i, j+1)$ and $(i+1, j)$

The foregoing specification introduces a nonstandard alternative. For the case of a disconnecting 2-AC, the possible assignment $j = i+2$ causes the added edge $(i+1, j)$ to be $(i+1, i+2)$, which is an edge of T. This violates the customary definition of an alternating cycle (where added edges must not belong to the tour subgraph). As will be seen however, it is a useful exception for the purpose of our development. In general, "exceptional" alternating paths embodied in ejection chains, which include constructions whose components are subpaths, provide a basis for useful heuristics and also for associated theorems about connectivity (Glover, 1992). (See Rego, 1996, for a highly efficient design and implementation of an ejection chain procedure for the TSP; related developments are also given in Zachariasen and Dam, 1996; Pesch and Glover, 1995; Rego and Roucairol, 1996; and Punnen and Glover, 1996.)

2.2. *Classes of 4-opt moves*

The class of 4-opt moves that has received special attention in the TSP literature is a conjunction of two disconnecting 2-ACs, arranged so that each reattaches the components of T that are disconnected by the other. Such moves are called "super moves" in Fiechter (1994) and "double-bridge" moves in Johnson and McGeoch (1996). We are interested in these and also in 4-opt moves from a second class whose members result by an appropriate combination of a disconnecting 2-AC and a connecting 2-AC. To identify those two types of 4-opt moves more precisely, we refer to the dropped edges of their two component ACs as $e(i1)$, $e(j1)$ and $e(i2)$, $e(j2)$, respectively, where we stipulate without loss of generality that $i1 < i2$. (No dropped edge of one AC can meaningfully duplicate a dropped edge of the other.) We then define the two component ACs to be *crossing* if $j1 < j2$ and *noncrossing* if $j1 > j2$. Six types of moves result from these definitions, which we list below for completeness.

- Type 1: Two disconnecting 2-ACs, which are crossing;
- Type 2: A disconnecting 2-AC and a connecting 2-AC, which are crossing;
- Type 3: Two disconnecting 2-ACs, which are noncrossing;
- Type 4: A disconnecting 2-AC and a connecting 2-AC, which are noncrossing;
- Type 5: Two connecting 2-ACs, which are crossing;
- Type 6: Two connecting 2-ACs, which are noncrossing.

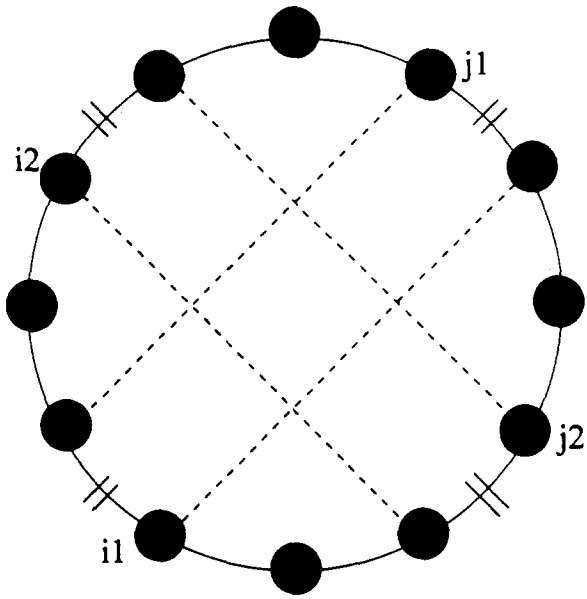

Figure 1. Type 1 move.

Type 1 moves constitute the so-called double-bridge moves and Type 2 moves, which are twice as numerous, can be viewed as replacing either one of the two "bridges" by a "twisted-bridge" whose edges cross each other. Figure 1 shows a Type 1 move, and Figure 2 shows a Type 2 move (which is one of the two Type 2 moves that are related to the Type 1 move of Figure 1). The tour edges that are dropped to create these moves are indicated as lines marked with bars, while the nontour edges that are added are indicated as dashed lines. The nodes $i1$, $j1$, $i2$, and $j2$, which determine the dropped edges $e(i1)$, $e(j1)$ and $e(i2)$, $e(j2)$, are also identified.

It is easy to determine that the moves of Types 3, 4, and 5 do not yield valid tours and hence are not of interest. The Type 6 move consists of two legitimate (2-opt) moves that can be performed independently. We therefore call Type 6 moves *decomposable* moves. (If the combination of the component moves is an improving move, at least one of these legitimate components is improving. If both components are improving we can choose them in succession.)

Type 1 and Type 2 moves are different, however, since they produce a valid tour and also contain a disconnecting component. Thus, while neither component of these moves may be a valid improving move, the complete move itself may be improving. Consequently, we call Type 1 and 2 moves *nondecomposable* moves. These observations lead to the following conclusions.

- *Remark 4.* Type 1 and Type 2 moves include all nondecomposable 4-opt moves except for (a subset of) the moves that qualify as connecting 4-ACs.

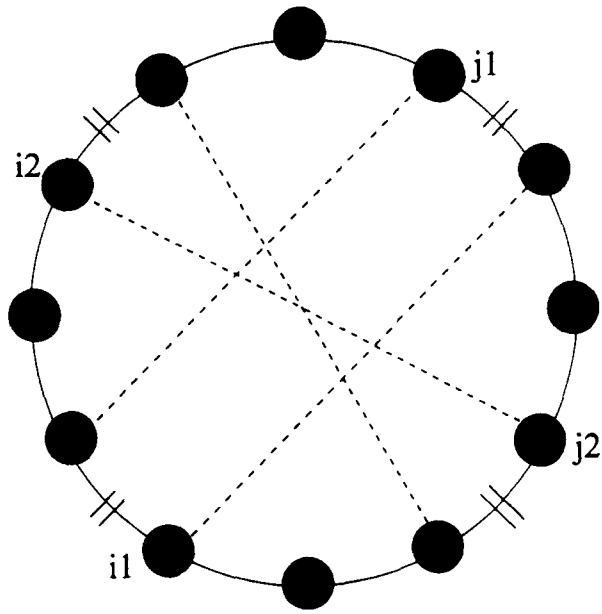

Figure 2. Type 2 move.

- *Remark 5.* If exactly one of the disconnecting components of a Type 1 or 2 move is a nonstandard 2-AC (if $j = i + 2$ for the dropped edges $e(i)$ and $e(j)$ of this component), then the resulting complete move is a 3-AC that is also a 3-opt move. (The "second copy" of the tour edge that is added by the nonstandard 2-AC is removed by the other 2-AC.) If both disconnecting components of a Type 1 move are nonstandard 2-ACs, then the resulting complete move is a 2-opt move (determined by the two dropped edges $e(i1)$ and $e(j2)$.)

Remark 4 provides a motivation for seeking improving and best instances of Type 1 and Type 2 moves, beyond the motivation provided by the fact that Type 1 moves are often included in effective heuristics. Remark 5 shows that, if we have a way of generating good Type 1 and Type 2 moves efficiently, we also reap a bonus of including a variety of 3-opt moves simultaneously. The number of such 3-opt moves is roughly four times the number of 2-opt moves. (There is no advantage to generating the special case 2-opt move of Remark 5 because 2-opt moves that may potentially qualify as good—for example, improving or best—are automatically generated in the course of identifying good Type 1 and Type 2 moves.)

3. An acyclic shortest path model

Our acyclic shortest path model for generating best (or improving) moves of Type 1 and Type 2 introduces a digraph S whose nodes correspond to *edge pairs* of T. More precisely,

S contains a source nodes s and a terminal node t, and remaining nodes of S identify edge pairs $e(i), e(j)$ of T as follows.

3.1. *Conventions underlying the construction of S*

1. The designation $x[i, j]$, where x is a variable term that takes the assignments $x = c$ (for "connecting") and $x = d$ (for "disconnecting"), corresponds to a 2-AC of type x that drops the two edges $e(i)$ and $e(j)$. The cost of such a 2-AC, which equals the sum of its added edges minus the sum of its dropped edges, is denoted cost_$x[i, j]$. (Hence, cost_$c[i, j]$ and cost_$d[i, j]$, respectively, denote the cost of a connecting 2-AC and a disconnecting 2-AC that drop $e(i)$ and $e(j)$.)

2. A class of nodes of S denoted reach_$x[i, j]$, for $i = 1$ to $n - 3$ and for $j = i + 2$ to $n - 1$ (where i and j take admissible values for $i1$ and $j1$ in the definitions of Type 1 and 2 moves), refers to the collection of 2-ACs of the form $x[p, j]$, $p = 1$ to i. The collection of all paths from the sources s to the node reach_$x[i, j]$ of S corresponds to the collection of all such 2-ACs, and the cost of a given path equals the cost of the corresponding move $x[p, j]$. (Thus, the shortest path from s to reach_$x[i, j]$ identifies the least cost 2-AC from this collection, and a predecessor trace identifies the value $p(\leq i)$ that yields this 2-AC.)

3. A class of nodes of S denoted cross_$x[i, j]$, for $i = 2$ to $n - 2$ and for $j = i + 2$ to n (where i and j take admissible values for $i2$ and $j2$ in the definitions of Type 1 and Type 2 moves), refers to the collection of all 2-ACs of the form $x[p, q]$ for $p = 1$ to $i - 1$ and for $q = i + 1$ to $j - 1$. (This is the collection of 2-ACs whose dropped edges can take the role of $e(i1)$ and $e(j1)$, and which cross the 2-AC whose dropped edges are $e(i2)$ and $e(j2)$, for $i = i2$ and $j = j2$.) The collection of all paths from s to cross_$x[i, j]$ corresponds to this collection of crossing 2-ACs, and the cost of a given path equals the cost of the corresponding $x[p, q]$. (Thus the shortest path to cross_$x[i, j]$ identifies the least-cost path from this collection, and a predecessor trace identifies both the values p and q that yields this 2-AC.)

4. Finally, S is organized so that arcs from nodes cross_$x[i, j]$ to the terminal node t generate precisely the set of paths from s to t that correspond to the union of Type 1 and Type 2 moves, and these arcs carry appropriate costs so that a shortest path from s to t identifies a least-cost move to Type 1 or 2 (inclusively).

The organization of S that achieves the outcomes indicated by the foregoing conventions is as follows.

3.2. *Structure of S*

For $x = c$ and $x = d$, and for $i = 1$ to $n - 3$ and $j = i + 2$ to $n - 1$, the digraph incorporates five types of arcs:

1. From s to reach_$x[i, j]$ with a cost of cost_$x[i, j]$,
2. From reach_$x[i - 1, j]$ to reach_$x[i, j]$ with a cost of 0,
3. From reach_$x[i, j]$ to cross_$x[i, j]$ with a cost of 0,
4. From cross_$x[i, j]$ to cross_$x[i, j + 1]$, with a cost of 0,

5. From cross_$x[i, j]$ to t with a cost of cost_$d[i + 1, j + 1]$, and an additional arc to t for $x = c$ with a cost of cost_$c[i + 1, j + 1]$.

Theorem. *The digraph S is acyclic and contains $O(n^2)$ arcs, and satisfies the properties asserted in conventions* 1 *through* 4.

Proof. The result follows directly from the specified structure of S and induction on i and j. □

The fact that S is acyclic permits a shortest path to be found by examining each arc exactly once and thus assures an $O(n^2)$ computation bound. There are a variety of ways to apply an acyclic shortest-path method to S, and we identify one way in particular that is conveniently organized for computer implementation. We will show that this method also is adapted to exploit strategies for shrinking S by considering only a limited number of candidates for dropped edges, to yield an $O(n)$ computation bound in place of $O(n^2)$.

3.3. Shortest-path method for S

Denote the cost of a shortest (currently known) path from s to the nodes reach_$x[i, j]$, cross_$x[i, j]$ and t by best_reach_$x[i, j]$, best_cross_$x[i, j]$ and best_t. Also denote the shortest path predecessors associated with these nodes (as a basis for recovering the identity of a shortest path) by pred_reach_$x[i, j]$, pred_cross_$x[i, j]$ and pred_t. The two latter predecessors will be maintained in an "aggregate" form, so that pred_t by itself yields all the information to identify a shortest path (and hence a best move).

There are three associated simple shortest-path updates, which provide the heart of the method. We present these updates in a special order for values of i and j that correspond to those specified by the algorithm to be described. An important consequence of executing the updates in this order is that all arrays indexed by $[i, j]$ can be collapsed to be indexed only by $[j]$, thus significantly reducing the array space required. (Note that cost_$x[i, j]$ is not an array but a definition based on dropping edges $e(i)$ and $e(j)$.) We include the relevant values of i in the following updates, to make visible the way in which the method traverses the digraph S. Nevertheless, only the j component of all arrays (excluding "cost arrays") needs to be considered during implementation.

Each update is executed for $i = 2$ to $n - 2$ and for $j = i + 2$ to n except as noted. The predecessor pred_cross_x is stored as an ordered pair, and pred_t is stored as two ordered pairs, pred_$t(1)$ and pred_$t(2)$. Also, we store with best_t a pair designated as "type". Cost terms that are duplicated in these updates of course need to be computed only once.

3.3.1. Best cross update (for $j \leq n - 1$)

If best_reach_$x[i - 1, j]$ < best_cross_$x[i - 1, j - 1]$, then
  best_cross_$x[i - 1, j]$ = best_reach_$x[i - 1, j]$
  pred_cross_$x[i - 1, j]$ = (pred_reach_$x[i - 1, j], j$)
else

best_cross_$x[i-1, j]$ = best_cross_$x[i-1, j-1]$
pred_cross_$x[i-1, j]$ = pred_cross_$x[i-1, j-1]$

3.3.2. Best reach update (for $j \leq n-1$)

If cost_$x[i, j]$ < best_reach_$x[i-1, j]$, then
  best_reach_$x[i, j]$ = cost_$x[i, j]$
  pred_reach_$x[i, j] = i$
else
  best_reach_$x[i, j]$ = best_reach_$x[i-1, j]$
  pred_reach_$x[i, j]$ = pred_reach_$x[i-1, j]$

(redundant when the array index i is suppressed)

3.3.3. Best terminal node update

For $y = c$ and $y = d$ when $x = d$, and for $y = d$ when $x = c$:
  If cost_$x[i, j]$ + best_cross_$y[i-1, j-1]$ < best_t, then
    best_t = cost_$x[i, j]$ + best_cross_$y[i-1, j-1]$
    pred_$t(1)$ = pred_cross_$y[i-1, j-1]$
    pred_$t(2) = (i, j)$
    type = (x, y)

Finally, to include the possibility that a simple 2-opt move may be best, we include the following.

3.3.4. Simple terminal node update

If cost_$c[i, j]$ < best_t, then
  best_t = cost_$c[i, j]$
  pred_$t(2) = (i, j)$
  type = (c, c)

3.4. Shortest path method

Except for the solution recovery, each step for the shortest-path method is executed for both $x = c$ and $x = d$.

- *Step 1.* (Initialization). best_$t = \infty$
  For $j = 3$ to $n - 1$:
    best_reach_$x[1, j]$ = cost_$x[1, j]$
    pred_reach_$x[1, j] = 1$
    Execute Simple Terminal Node Update for $i = 1$
  end

- *Step 2.* (Main Step).
  For $i = 2$ to $n - 2$
  best_cross_$x[i-1, i+1]$ = best_reach_$x[i-1, i+1]$
  pred_cross_$x[i-1, i+1]$ = (pred_reach_$x[i-1, i+1], i+1$)
  For $j = i + 2$ to n:
  Execute, in sequence
  Best Cross Update ($j \leq n - 1$)
  Best Reach Update ($j \leq n - 1$)
  Best Terminal Node Update
  Simple Terminal Node Update
  end
  end
- *Step 3.* (Solution Recovery).
  best_t gives the cost of the best move.
  (1) If type $= (c, c)$, the best move is a simple $c[i, j]$ (2-opt) move for $(i, j) =$ pred_$t(2)$.
  (2) Otherwise, for type $= (x, y)$ (excluding $x = y = c$), the best move is composed of the component move $y[i1, j1]$ for $(i1, j1) =$ pred_$t(1)$ and the component move $x[i2, j2]$ for $(i2, j2) =$ pred_$t(2)$.

In accordance with our previous comments, the arrays can be reduced so that $[1, j]$ in Step 1 can be replaced by $[j]$ (except in reference to cost_$x[1, j]$). Similarly $[i - 1, i + 1]$ in Step 2 can be replaced by $[i + 1]$. Unlike the other updates, the simple terminal node update does not have to be executed in the sequence indicated.

To organization of the method makes it clear that the computation required to find a best move is a simple multiple of that required to find a best 2-opt move (which occurs by retaining only the simple terminal node update).

4. An accelerated $O(n)$ method

The digraph S can be collapsed, allowing the foregoing method to be simplified, by restricting attention to a fixed (or bounded) number of edges of T as candidates to become the dropped edges $e(i)$ and $e(j)$. The determination of such edges can be carried out by a variety of candidate list strategies (e.g., Glover, 1989; Glover and Laguna, 1996). (A straightforward strategy in the present setting, for example, is to periodically evaluate all or a significant number of tour edges as candidates to be dropped, based on the quality of the moves to which they contribute, and then to restrict attention for some iterations to a subset of those that received highest evaluations.)

To implement the method under this form of restriction, suppose that the candidate edges to be dropped are indexed as $e(p(h))$ for $h = 1$ to q, where $p(h)$ increases with increasing values of h, and $p(1) = 1$. Then the shortest-path method successfully finds best Type 1 and Type 2 moves over the indicated collection of dropped edges by the following changes:

- Step 1: $[1, n]$ becomes $[1, p(q)]$
  $j = 3$ to $n - 1$ becomes $j = p(h), h = 3$ to $q - 1$

- Step 2: $i = 2$ to $n - 2$ becomes $i = p(h)$, $h = 2$ to $q - 2$
  $i + 1$ becomes $p(h + 1)$
  $j = i + 2$ to n becomes $j = p(k)$ for $k = h + 2$ to q
  $j - 1$ in the updates becomes $p(k - 1)$.

The preceding organization for finding best Type 1 and Type 2 moves may skip some (i, j) combinations that give legitimate 2-opt moves. These 2-opt moves can also be checked by applying the simple terminal node update under the following additional conditions:

- Step 1: $j = p(2)$ if $p(2) \geq 3$
- Step 2: $j = p(h + 1)$ if $p(h + 1) \geq p(h) + 2$ (accompanied by $h = q - 1$ in setting $i = p(h)$, if $p(q) \geq p(q - 1) + 2$).

The foregoing changes then reduce the effort to $O(q^2)$ for the chosen constant q. The overall effort is $O(n)$ due to the overhead of maintaining the sequential indexing. Together with this, or separately, one can impose thresholds on costs of chosen partial add/drop components of moves to limit the moves considered for full evaluation.

5. Asymmetric problems

The foregoing procedures can readily by adapted to asymmetric TSPs. If attention is restricted to Type 1 moves, then the tour orientation remains unaffected and the adaptation is trivial. In the case of Type 2 moves, the process of sequentially indexing the nodes of T can be accompanied by generating a cumulative forward cost $F(i)$, for each arcs $a(i) = (i, i+1)$, which equals the sum of the costs of arcs $a(h)$ for $h = 1$ to i, with $F(0) = 0$. Similarly, a cumulative reverse cost $R(i)$ is generated for each arc $a'(i) = (i + 1, i)$, which equals the sum of the costs of arcs $a'(h)$ for $h = i$ to n, with $R(n + 1) = 0$. The cost of inverting a path consisting of arcs $a(p)$ to $a(q)$ for $p \leq q$ (replacing these arcs with $a'(q)$ to $a'(p)$), is given by $R(p) - R(q + 1) - (F(q) - F(p - 1))$. Appropriate move evaluations can then easily be identified, incurring an overall increase in computational overhead of $O(n)$, which yields the same order bound as for the symmetric problem.

6. Conclusions

The emphasis on speed in current TSP implementations suggests the merit of giving special attention to accelerated versions of the method. For this purpose, candidate list strategies for identifying selected subsets of dropped edges may be enhanced by the use of memory-based strategies of tabu search (see, e.g., Hertz, Taillard, and de Werra, 1996; Glover and Laguna, 1996). In particular, tabu search intensification strategies, which systematically bias the search to incorporate attributes of previous high-quality solutions within current solutions, can be applied to identify edges and subpaths to be assigned preferred status for being included in tours. Consequently, such strategies implicitly identify remaining edges of T as appropriate candidates to be dropped. (Probabilistic variations of these strategies result by mapping evaluations of the relative attractiveness of being included and excluded directly

into probabilities for choosing edges to be removed.) Similarly, nontour edges that are among those preferred for inclusion implicitly identify tour edges (adjacent to the nontour edges) that deserve to be among the preferred candidates to be dropped. Diversification strategies of tabu search, which drive the search to generate configurations that are atypical relative to the search history, similarly identify edges that are preferable to be added and dropped and can be applied in a corresponding fashion.

Finally, we observe that the approaches of this article have a natural role in creating a "coordinated neighborhood" design. Experience shows that TSP methods often benefit from implementing more than a single type of move. Acyclic shortest-path models such as the one for finding good 4-opt moves (and related ejection chain models for finding other types of good moves) offer a chance to develop better "multimove" strategies.

Notably, these approaches embody an unconventional way to integrate optimization with heuristics. Instead of incorporating heuristic rules and trial solution procedures within an optimization method to improve its performance, the scheme is reversed to embed special optimization models within a heuristic to render it more effective.

The special shortest-path structures we have introduced to achieve this strategic design have additional uses. In a sequel we show how to link the constructions of this article to complimentary ejection chain models to yield best moves over more complex sets of alternatives.

References

Berge, C. (1962). *Theory of Graphs and Its Applications*. London: Methuen.

Fiechter, C.-N. (1994). "A Parallel Tabu Search Algorithm for Large Traveling Salesman Problems." *Discrete Applied Math* 51, 243–267.

Glover, F. (1989). "Candidate List Strategies and Tabu Search." CAAI Research Report, University of Colorado, Boulder, July.

Glover, F. (1992). "Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems." University of Colorado. Shortened version published in *Discrete Applied Mathematics* (1996), 65, 223–253.

Glover, F., and M. Laguna. (1996). *Tabu Search*. Boston: Kluwer.

Hertz, A., É. Taillard, and D. de Werra (1996). "Tabu Search." In E.H.L. Aarts, and J.K. Lenstra (Eds.), *Local Search in Combinatorial Optimization*. Chichester: Wiley.

Johnson, D.S., and L.A. McGeoch. (1996). "The Traveling Salesman Problem: A Case Study in Local Optimization." In E.H.L. Aarts, and J.K. Lenstra (eds.), *Local Search in Combinatorial Optimization*. New York: Wiley.

Martin, O., S.W. Otto, and E.W. Felten. (1992). "Large-Step Markov Chains for TSP Incorporating Local Search Heuristics." *Operations Research Letters* 11:219–224.

Pesch, E., and F. Glover. (1995). "TSP Ejection Chains." Graduate School of Business, University of Colorado, Boulder. To appear in *Discrete Applied Mathematics*.

Punnen, A., and F. Glover, (1996). "Implementing Ejection Chains with Combinatorial Leverage for TSPs." Graduate School of Business, University of Colorado, Boulder.

Rego, C. (1996). "Relaxed Tours and Path Ejections for the Traveling Salesman Problem." Technical report, Laboratoire PRiSM, Université de Versailles.

Rego, C., and C. Roucairol. (1996). "A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem." In I.H. Osman and J.P. Kelly (Eds.), *Meta-Heuristics: Theory & Applications* (pp. 661–675). Boston: Kluwer.

Zachariasen, M., and M. Dam. (1996). "Tabu Search on the Geometric Traveling Salesman Problem." In I.H. Osman, and J.P. Kelly (Eds.), *Meta-Heuristics: Theory & Applications* (pp. 571–587). Boston: Kluwer.