NEW INSERTION AND POSTOPTIMIZATION PROCEDURES FOR THE TRAVELING SALESMAN PROBLEM

MICHEL GENDREAU
University of Montreal, Montreal, Canada

ALAIN HERTZ
École Polytechnique Fédérale de Lausanne, Switzerland

GILBERT LAPORTE
University of Montreal, Montreal, Canada

(Received September 1990; revision received October 1991; accepted February 1992)

This paper describes a new insertion procedure and a new postoptimization routine for the traveling salesman problem. The combination of the two methods results in an efficient algorithm (**GENIUS**) which outperforms known alternative heuristics in terms of solution quality and computing time.

The Traveling Salesman Problem (TSP) is one of the most widely studied problems in combinatorial optimization. It may be defined as follows. Let $G = (V, A)$ be a graph, where $V = \{v_1, \dots, v_n\}$ is the vertex set and $A = \{(v_i, v_j): v_i, v_j \in V\}$ is the arc set. With each arc (v_i, v_j) is associated a nonnegative cost c_{ij}. The TSP consists of finding the least cost circuit passing through every vertex exactly once. Such a circuit is said to be *Hamiltonian*. The problem is *symmetric* if and only if $c_{ij} = c_{ji}$ for all i, j. It satisfies the *triangle inequality* if and only if $c_{ij} + c_{jk} \geq c_{ik}$ for all i, j, k; a particular case is that of *Euclidean* problems whose costs correspond to Euclidean distances.

The most common interpretation of the TSP is that of a salesman seeking to determine a shortest Hamiltonian circuit through n cities. A number of Vehicle Routing Problems (VRPs) also can be derived from the TSP. One of the vertices is then designated as a *depot* and a number of least-cost vehicle routes must be designed from the depot to the remaining vertices, subject to side constraints. It is often possible to solve VRPs by adapting TSP algorithms (e.g., Laporte, Nobert and Desrochers 1985, Laporte, Mercure and Nobert 1986). More recently, new TSP applications have been described in several other fields such as drilling of printed circuit boards (Grötschel, Jünger and Reinelt 1989, Reinelt 1992), sequencing of punch operations in a flexible manufacturing cell (Chauny et al. 1987), and crystallography (Bland and Shallcross 1989).

Several of these applications are large scale and can rarely be tackled by means of exact algorithms. Some notable exceptions are the drilling problems solved by Padberg and Rinaldi (1990) using a polytopal cutting plane algorithm. Most of these problems require fast, low complexity heuristics for their solution. The objective of this paper is to describe such a heuristic which outperforms previously published methods on test problems. It includes an improved insertion procedure and a new postoptimization routine which can also be used in conjunction with other algorithms.

The remainder of this paper is organized as follows. In Section 1, we briefly review a number of TSP heuristics relevant to this paper. The insertion procedure (**GENI**) and the improvement method (**US**) are described in the next two sections. Computational results, reported in Section 4, indicate that the combination of these two heuristics (**GENIUS**) compares favorably with other algorithms. The conclusion follows in Section 5.

1. HEURISTIC ALGORITHMS FOR THE TSP

Broadly speaking, TSP heuristics can be classified into *tour construction procedures*, which consist of gradually building a solution by adding a new vertex at each step, and *tour improvement procedures*, which improve upon a feasible solution by performing various exchanges. The best methods are *two-phase procedures* that combine these two features.

Subject classification: Networks/graphs, traveling salesman: insertion and postoptimization procedures.
Area of review: DISTRIBUTION, TRANSPORTATION AND LOGISTICS.

0030-364X/92/4006-1086 $01.25

Tour construction methods include the well known nearest neighbor heuristic and a number of *insertion procedures* that use various criteria (Rosenkrantz, Stearns and Lewis 1977). For empirical comparisons and asymptotic analyses of these heuristics, see Golden and Stewart (1985) and Ong and Huang (1989). Some authors have also proposed linear or near-linear time heuristics applicable to large-scale TSPs arising from manufacturing contexts (see, for example, Bartholdi and Platzman 1988, and Reinelt 1992). These methods are meant to quickly provide good solutions, often in real time. However, as far as solution quality is concerned, they are usually not the most refined algorithms available.

Classical improvement procedures are those of Lin (1965), Lin and Kernighan (1973), and Or (1976). As shown by Stewart (1987), such algorithms often can be accelerated by limiting the number of exchange candidates.

In addition, a number of global optimization techniques have also been adapted for the TSP. The first, *simulated annealing*, is derived from the early work of Metropolis et al. (1953) and was first applied to the field of combinatorial optimization by Kirkpatrick, Gelatt and Vecchi (1983), but its application to the TSP seems to have produced mixed results (Soriano 1989). The second of these methods, *tabu search*, is due to Glover (1977) and appears to have more potential for the TSP. Both simulated annealing and tabu search can be termed open-ended improvement methods. They are not the most appropriate techniques for quickly generating good tours and their performance is directly related to running time.

One of the better two-phase methods is the so-called CCAO heuristic of Golden and Stewart (1985). It was designed for (planar) symmetric Euclidean problems and it exploits their geometrical properties. This algorithm constructs an initial tour consisting of the convex hull of vertices. Vertices not yet on the tour are gradually included by first considering all possible insertions, and then selecting the best move according to a largest angle criterion. The tour thus obtained is then improved by means of the Or-opt postoptimization procedure.

In spite of their good performance, two-phase methods such as CCAO suffer from two serious drawbacks. First, several unpromising moves are considered: this occurs whenever an attempt is made to insert a vertex between two vertices distant from it. Second, the insertion phase of these methods is myopic in the following sense. Since insertions are executed sequentially without much concern for global optimality, they may result in a succession of bad decisions that the postoptimization phase will be unable to undo. The algorithm we are proposing attempts fewer insertions, but executes each one more carefully by performing a limited number of local transformations of the tour, simultaneously with the insertion itself.

2. A GENERALIZED INSERTION PROCEDURE (GENI)

The principles behind our Generalized Insertion Procedure (**GENI**) apply equally well to symmetric and asymmetric problems. However, in order to simplify the presentation, and since our computational results are from symmetric problems only, we have chosen to restrict our description to this case. Extending **GENI** to asymmetric problems is straightforward.

The main feature of **GENI** is that insertion of a vertex v in the tour does not necessarily take place between two vertices which are consecutive when they are first considered. However, after insertion, these two vertices become adjacent to v. Suppose that we wish to insert v between any two vertices v_i and v_j of the tour. For a given orientation of the tour, let v_k be a vertex on the path from v_j to v_i, and v_l be a vertex on the path from v_i to v_j. For any vertex v_h on the tour, let v_{h-1} be its predecessor and v_{h+1} its successor. Insertion of v between v_i and v_j can be done in one of two ways, as illustrated in Figures 1 and 2.

2.1. Type I Insertion

Here $v_k \neq v_i$ and $v_k \neq v_j$. Inserting v in the tour results in the deletion of arcs (v_i, v_{i+1}), (v_j, v_{j+1}), and (v_k, v_{k+1}) and in their replacement by (v_i, v), (v, v_j), (v_{i+1}, v_k), and (v_{j+1}, v_{k+1}). This implies that the two paths $(v_{i+1}, \ldots, v_j)$ and $(v_{j+1}, \ldots, v_k)$ are reversed.

Note that setting $j = i + 1$ and $k = j + 1$ yields the standard insertion procedure. Moreover, for a given

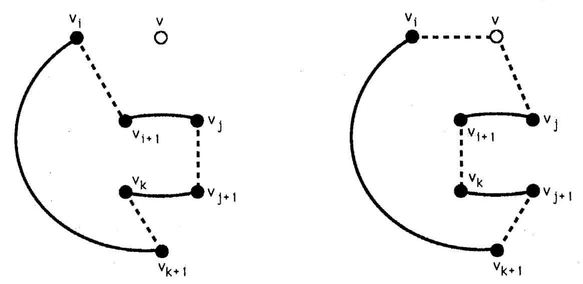


Figure 1. Type I insertion of vertex v between v_i and v_j.

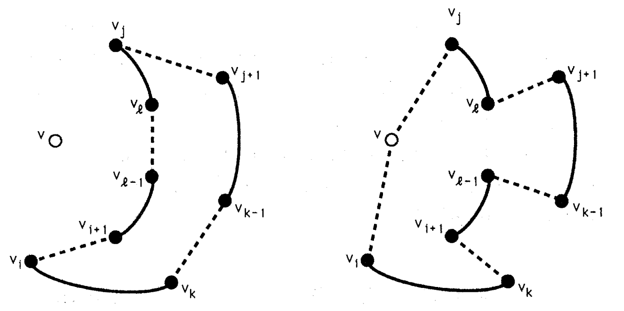


Figure 2. Type II insertion of vertex v between v_i and v_j.

choice of v_i, the family of possible insertions coincides with that explored in the first step of the dynamic 3-opt heuristic of Steiglitz and Weiner (1968).

2.2. Type II Insertion

Here $v_k \neq v_j$ and $v_k \neq v_{j+1}$; $v_l \neq v_i$ and $v_l \neq v_{i+1}$. Inserting v in the tour results in the deletion of (v_i, v_{i+1}), (v_{l-1}, v_l), (v_j, v_{j+1}), and (v_{k-1}, v_k). These arcs are replaced by (v_i, v), (v, v_j), (v_l, v_{j+1}), (v_{k-1}, v_{l-1}), and (v_{i+1}, v_k). As above, the paths $(v_{i+1}, \ldots, v_{l-1})$ and $(v_l, \ldots, v_j)$ are reversed.

The **GENI** algorithm considers the two possible orientations of the tour for each possible insertion. Since the potential number of choices for v_i, v_j, v_k, v_l is on the order of n^4, we limit the search as follows. For any vertex $v \in V$, define its p-neighborhood $N_p(v)$ as the set of the p vertices on the tour closest to v (with respect to the c_{ij}'s); if v has fewer than p neighbors on the tour, they all belong to $N_p(v)$. Then, for a given parameter p, we first select $v_i, v_j \in N_p(v)$, $v_k \in N_p(v_{i+1})$, and $v_l \in N_p(v_{j+1})$. We also consider all insertions of v between two consecutive vertices v_i and v_{i+1}, as long as $v_i \in N_p(v)$. In practice, p is a relatively small number. The **GENI** algorithm may now be described.

GENI Algorithm

Step 1. Create an initial tour by selecting an arbitrary subset of three vertices. Initialize the p-neighborhoods of all vertices.

Step 2. Arbitrarily select a vertex v not yet on the tour. Implement the least cost insertion of v considering the two possible orientations of the tour and the two insertion types. Update the p-neighborhoods of all vertices to account for the fact that v is now on the tour.

Step 3. If all vertices are now part of the tour, stop. Otherwise go back to Step 2.

The complexity of this procedure is determined as follows. In Step 2, $O(p^4)$ choices of v_i, v_j, v_k and v_l must be considered. For the best choice, v is inserted in the tour which is then updated: This requires $O(n)$ time. Since Step 2 is executed $n - 3$ times, the overall complexity of **GENI** is $O(np^4 + n^2)$.

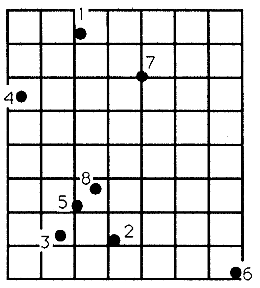
The interest of the proposed approach lies in the integration of local optimization steps within an insertion procedure. Similar ideas have been put forward previously by Steiglitz and Weiner. In essence, their dynamic 3-opt method executes a sequence of moves consisting of the best simple insertion followed by a 3-opt search. In **GENI**, type I insertions are equivalent to selecting the best of several moves, each consisting of a simple insertion followed by only one 3-opt exchange. Type II insertions use one 4-opt exchange instead. What makes our method efficient is the fact that the number of potential insertions is restricted to the neighborhood of the vertex to be added to the tour. Also, the improvement search is limited to the most promising moves because the reconnecting arcs link vertices that are close to one another.

It is worth noting that even in Euclidean problems, inserting a new vertex in the tour sometimes results in a lesser cost. This never happens in standard insertion procedures. To illustrate, consider the example depicted in Figure 3. Here $p = 4$ and vertices are inserted in the tour in numerical order. An interesting phenomenon occurs when vertex 6 is inserted: The procedure then produces a tour which crosses itself. This is so because the best available insertion, between vertices 1 and 2, is infeasible because 1 does not belong to $N_4(6)$. Also, a cost reduction from 238.6 to 229.0 occurs when vertex 8 is inserted. Note that the final solution is not optimal. Indeed, the best tour (1, 7, 6, 2, 3, 5, 8, 4, 1) has a cost of 223.0 and can be reached with $p = 5$.

3. UNSTRINGING AND STRINGING (US)

In addition to **GENI**, we have also developed a post-optimization algorithm which consists of removing a vertex from a feasible tour and of inserting it back. By analogy with bead stringing, these processes are called *Unstringing* and *Stringing* (**US**). This procedure can be applied to tours produced by any algorithm. The stringing procedure is identical to Step 2 of **GENI**. The unstringing procedure is just the reverse. When a vertex v_i is removed, we consider two ways of reconnecting the tour. These are illustrated in Figures 4 and 5.

vertices	coordinates x	y
1	21.8	72.8
2	31.5	10.8
3	16.3	13.7
4	3.7	54.8
5	21.0	21.7
6	69.3	1.3
7	40.8	60.1
8	27.8	26.2



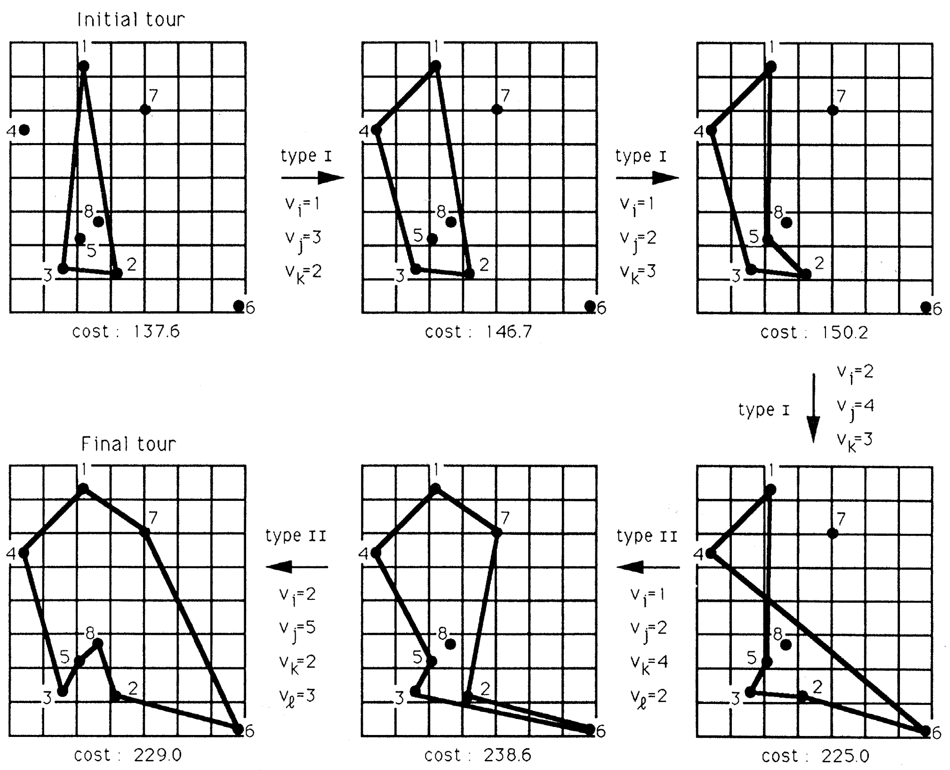


Figure 3. Illustration of the **GENI** algorithm on an 8-vertex problem.

3.1. Type I Unstringing

Let $v_j \in N_p(v_{i+1})$, and for a given orientation of the tour let $v_k \in N_p(v_{i-1})$ be a vertex on the path $(v_{i+1}, \ldots, v_{j-1})$. Then arcs (v_{i-1}, v_i), (v_i, v_{i+1}), (v_k, v_{k+1}), and (v_j, v_{j+1}) are deleted, while arcs (v_{i-1}, v_k), (v_{i+1}, v_j), and (v_{k+1}, v_{j+1}) are inserted. Also, the two paths $(v_{i+1}, \ldots, v_k)$ and $(v_{k+1}, \ldots, v_j)$ are reversed.

3.2. Type II Unstringing

Let $v_j \in N_p(v_{i+1})$. For a given orientation of the tour, let $v_k \in N_p(v_{i-1})$ be a vertex on the path $(v_{j+1}, \ldots, v_{i-2})$; also let $v_l \in N_p(v_{k+1})$ on the path $(v_j, \ldots, v_{k-1})$. Then arcs (v_{i-1}, v_i), (v_i, v_{i+1}), (v_{j-1}, v_j), (v_l, v_{l+1}), and (v_k, v_{k+1}) are deleted, while arcs (v_{i-1}, v_k), (v_{l+1}, v_{j-1}), (v_{i+1}, v_j), and (v_l, v_{k+1}) are inserted. Also, the two paths $(v_{i+1}, \ldots, v_{j-1})$ and $(v_{l+1}, \ldots, v_k)$ are reversed.

The US algorithm may now be described.

Algorithm US

Step 1. Consider an initial tour τ of cost z. Set $\tau^* := \tau$, $z^* := z$ and $t := 1$.

Step 2. Starting from tour τ, apply the unstringing and stringing procedures with vertex v_t, considering in each case the two possible types of operation and the two-tour orientations. Let τ' be the tour obtained and let z' be its cost. Set $\tau := \tau'$ and $z := z'$.

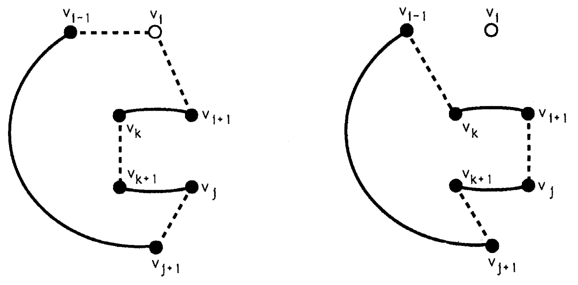


Figure 4. Type I unstringing of vertex v_i from the tour.

- If $z < z^*$, set $\tau^* := \tau$, $z^* := z$ and $t := 1$; repeat Step 2.
- If $z \geqslant z^*$, set $t := t + 1$.
- If and $t = n + 1$, stop: the best available tour is τ^* and its cost is equal to z^*. Otherwise, repeat Step 2.

Note that in the spirit of the simulated annealing and tabu search methods, Step 2 is applied to the last, but not necessarily to the best tour available, since the costs of two consecutive tours produced by Step 2 may increase. Indeed, a vertex to be removed from the tour is not necessarily located between two vertices belonging to its p-neighborhood. Thus, reinserting a vertex in the position it occupied before its removal from the tour may not be allowed. However, the best known tour is always stored.

4. COMPUTATIONAL RESULTS

To assess the efficiency of **GENI** and **US**, and to make comparisons with existing methods, we have carried out several series of computational tests both on randomly generated problems and on TSPs described in the operations research literature. The first set of problems were derived by generating n points in

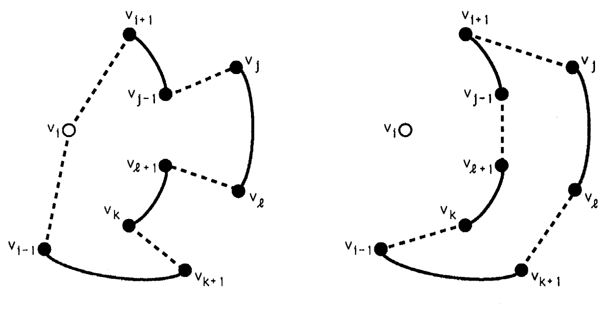


Figure 5. Type II unstringing of vertex v_i from the tour.

$[0, 100]^2$, according to a continuous uniform distribution, and by then computing the symmetric Euclidean distances (costs) between these points.

The various procedures used in the computational tests are:

C:	the convex hull procedure in Step 1 of the **CCAO** algorithm (Golden and Stewart);
CA:	the insertion phase (Steps 2, 3 and 4) of **CCAO**;
O:	the Or-opt algorithm (Or 1976);
GENI:	the proposed generalized insertion algorithm; and
US:	the unstringing and stringing routines.

Several combinations of these procedures were tested: **CCA, CCAO, GENI, GENIO** and **GENIUS**. In addition, we have tested a modified version of **GENI** (**CGENI**) which consists of using the convex hull of vertices as an initial tour. The various versions of **GENI** were tested for $p = 2, \ldots, 7$. The convex hull procedure (**C**) is the $O(n \ln n)$ GRAHAMHULL routine without polar coordinates, described in Preparata and Shamos (1985, pp. 108–109). For **CCA** and **O**, we have programmed a careful and efficient implementation of the algorithms, as described by their respective authors. All codes were written in Pascal and tests were executed on a SUN SPARC Workstation. The first set of computational results are reported in Table I. For the various algorithms that were compared, we provide the average solution costs (in column Cost) and the average computation times in seconds (in column Time) over 100 independently generated problems.

A number of observations can be derived from these results. First, comparing **GENI** and **CGENI** shows that these two procedures are more or less equivalent to one another, both in terms of the objective function and of computing speed. In other words, whether the generalized insertion routine is initialized from scratch or from the convex hull is immaterial. Similar comparisons were made between **GENIO** and **CGENIO** as well as between **GENIUS** and **CGENIUS** and in both cases the same conclusion was reached. Not surprisingly, for **GENI, GENIO** and **GENIUS**, solution quality usually improves as p becomes larger. While computing times almost always increase with p for **GENI** and **GENIUS**, this is not the case for **GENIO**: Here they attain their minimum at an intermediate value of p. Our proposed postoptimization procedure (**US**) is better than Or-opt (**O**), when executed after **GENI**. This can be seen by comparing **GENIO** and **GENIUS**: When $p \geqslant 3$, the

Table I
Average Solution Costs and Computation Times for Random Euclidean Problems Over 100 Trials

		$n = 100$		$n = 200$		$n = 300$		$n = 400$		$n = 500$	
Procedure	p	Cost	Time	Cost	Time	Cost	Time	Cost	Time	Cost	Time
GENI	2	842.5	0.4	1176.3	1.4	1431.2	3.1	1644.9	5.4	1838.8	8.4
	3	815.2	0.5	1134.1	1.7	1382.1	3.5	1583.2	6.0	1770.4	9.2
	4	807.3	0.8	1121.6	2.3	1364.8	4.5	1566.6	7.4	1748.5	11.0
	5	803.7	1.4	1115.3	3.6	1355.3	6.6	1557.7	10.1	1737.7	14.4
	6	799.1	2.6	1111.1	6.1	1351.8	10.4	1549.6	15.2	1728.8	20.9
	7	798.1	4.5	1109.1	10.2	1348.7	16.7	1547.5	23.8	1724.9	31.8
CGENI	2	842.3	0.4	1183.1	1.5	1436.0	3.2	1646.5	5.6	1839.4	8.6
	3	813.7	0.5	1134.6	1.7	1383.6	3.6	1585.4	6.1	1768.0	9.3
	4	804.6	0.8	1121.6	2.3	1365.4	4.6	1565.2	7.5	1746.2	11.1
	5	800.8	1.4	1114.7	3.6	1354.8	6.6	1556.2	10.2	1734.1	14.5
	6	799.4	2.5	1110.0	6.0	1351.6	10.2	1549.8	15.1	1727.4	20.8
	7	797.5	4.4	1107.7	10.0	1348.1	16.4	1546.9	23.5	1724.9	31.3
GENIO	2	809.5	2.5	1126.7	16.5	1369.9	52.0	1576.4	117.9	1756.5	200.9
	3	803.2	1.8	1117.6	9.9	1358.0	30.7	1556.3	65.9	1739.6	106.8
	4	799.8	1.8	1111.7	8.7	1352.2	23.2	1550.5	50.5	1730.6	79.9
	5	799.4	2.3	1108.0	8.9	1347.0	21.8	1547.2	43.4	1725.1	67.4
	6	796.1	3.4	1105.0	11.0	1345.4	23.6	1542.2	43.2	1719.4	65.8
	7	794.9	5.4	1104.0	14.6	1343.1	29.5	1540.6	50.3	1716.5	72.8
GENIUS	2	812.0	2.1	1130.7	9.4	1379.9	22.2	1582.9	43.2	1769.6	67.2
	3	796.8	2.5	**1107.6**	**10.0**	**1347.0**	**23.5**	**1545.4**	**40.8**	**1725.0**	**70.0**
	4	791.8	3.4	**1100.1**	**11.8**	**1337.1**	**26.2**	**1535.5**	**46.0**	**1710.7**	**71.1**
	5	791.2	5.4	1096.7	15.9	**1332.5**	**33.3**	**1531.9**	**54.2**	**1704.3**	**82.4**
	6	788.3	8.7	1095.3	23.8	**1332.2**	**45.4**	**1528.1**	**70.9**	**1701.0**	**109.1**
	7	788.6	13.6	1095.6	35.6	1330.6	66.7	1526.9	107.2	**1698.6**	**142.4**
CCA		820.0	0.7	1149.4	3.5	1405.6	8.6	1619.1	16.2	1810.9	27.3
CCAO		798.1	2.1	1111.0	14.9	1354.8	48.5	1556.8	112.1	1735.3	196.6

latter algorithm always yields better results than **GENIO**, although it is slower for $p \geqslant 5$. Similarly, comparing the two procedures that do not use postoptimization (**GENI** and **CCA**) shows that **GENI** is always better than **CCA** provided that $p \geqslant 3$, and quicker when p is not too large. Also, when $n \geqslant 200$, using our generalized insertion procedure without postoptimization (**GENI**) with a large enough value of p appears to be faster and better than **CCAO**.

Overall, the algorithm that yields the best solutions is **GENIUS** with $p \geqslant 3$. When $n \geqslant 200$, there always exists a range of values of p for which **GENIUS** is at the same time better and faster than **CCAO**; the corresponding results are indicated in boldfaced numbers. The average ratios of the solution values reported in Table II indicate that the improvement provided by **GENIUS** over **CCAO** increases with problem size, and is equal to approximately 2% in the best cases.

Table II
Solution Value and Computation Time Average Ratios: **GENIUS** ÷ **CCAO**

	$n = 100$		$n = 200$		$n = 300$		$n = 400$		$n = 500$	
p	Cost	Time	Cost	Time	Cost	Time	Cost	Time	Cost	Time
2	1.017	1.00	1.018	0.63	1.019	0.46	1.017	0.38	1.020	0.34
3	0.998	1.67	**0.997**	**0.67**	**0.994**	**0.48**	**0.992**	**0.36**	**0.994**	**0.36**
4	0.992	1.62	**0.990**	**0.79**	**0.987**	**0.54**	**0.986**	**0.41**	**0.986**	**0.36**
5	0.991	2.57	0.987	1.07	**0.984**	**0.69**	**0.984**	**0.48**	**0.982**	**0.42**
6	0.988	4.14	0.986	1.60	**0.983**	**0.94**	**0.982**	**0.63**	**0.980**	**0.55**
7	0.988	6.48	0.986	2.39	0.982	1.38	**0.981**	**0.96**	**0.979**	**0.72**

This indicates that the solutions produced by **GENIUS** may be very close to optimal in view of the fact that on 14 out of 15 problems tested by Golden and Stewart, **CCAO** yielded a solution having a cost within 2% of the best known cost (in five cases, all problems with $n = 100$, the best available solution was also known to be optimal).

The selection of p is an interesting question and the answer lies, of course, in the tradeoff one is willing to make between solution quality and computing time. For the values of n considered, and given the low solution times obtained, using as large a value of p as possible seems a reasonable choice, although there are diminishing returns. If computation time is important, an attractive solution could be to use **GENI** with a large p and not execute **US**. For example, **GENI** with $p = 7$ provides a solution value increase of about 1% with respect to **GENIUS**, for one quarter of the computing time.

In addition to these tests on randomly generated problems, we have compared **GENI** and **GENIUS** with other algorithms on the following problems, where alternative heuristics had already been tested and for which optimal values were available:

K24:	a 100-vertex problem (number 24) described in Krolak, Felts and Marble (1971);
GRID100:	a 100-vertex grid problem generated as in Çerny (1985);
GRID400:	a 400-vertex problem generated in the same fashion;
G442:	the Grötschel, Jünger and Reinelt 442-vertex problem; and
P532:	the Padberg and Rinaldi (1987) 532-vertex problem.

All problems were solved with **GENI**, **GENIUS**, **CCA** and **CCAO**. For these, computation times are for the SUN SPARC Workstation. We report solution values and computing times for these problems, when available, obtained with Lin's 2-opt algorithm (Reinelt 1992), with the Lin and Kerninghan algorithm (Reinelt 1992), with simulated annealing (Rossier, Troyon and Liebling 1986, Troyon 1988), and with tabu search (Fiechter 1990). These results are reported in Table III, along with the various computers used.

Table III
Computational Results for Problems Described in the Operations Research Literature

	Problem														
	K24			GRID100			GRID400			G442			P532		
Procedure	p	Cost	Time	p	Cost	Time	p	Cost	Time	p	Cost	Time	p	Cost	Time
GENI	5	21,792	1.7	3	100	0.6	3	400	7.5	5	52.77	14.0	5	28,433	18.8
GENIUS	5	21,282	4.7	3	100	2.6	3	400	23.5	5	51.27	114.3	5	28,175	95.4
CCA		21,512	0.8		106.2	0.5		471.6	11.0		55.25	23.2		29,559	35.7
CCAO		21,320	2.0		102.5	1.4		419.1	151.8		52.79	117.7		28,726	277.4
2-Opt					103.3	18.0[a]					54.67	29.6[a]			
Lin-Kernighan											52.54	46.1[a]			
Simulated Annealing					100	22.3[b]					51.42	238.0[c]		28,418[d]	448.0[c]
Tabu Search											51.45[e]	254.0[h]		28,090[e]	267.0[h]
											51.10[f]			27,839[f]	
											51.94[g]			28,422[g]	
Optimal Value		21,282[i]			100[j]			400[j]			50.67[k]			27,686[l]	

[a] Reinelt (1992); SUN 3/60 Workstation.
[b] Rossier, Troyon and Liebling (1986); CYBER 170-835.
[c] Troyon (1988); VAX 8600.
[d] P532 problem with truncated distances: The optimal solution value for the truncated version is 27,424.
[e] Fiechter (1990); average solution value over 100 trials.
[f] Fiechter (1990); best solution value over 100 trials.
[g] Fiechter (1990); worst solution value over 100 trials.
[h] Fiechter (1990); Silicon Graphics Station (1.6 Mflops); average running time over 100 trials.
[i] Crowder and Padberg (1980).
[j] Çerny (1985).
[k] Grötschel, Jünger and Reinelt (1989).
[l] Padberg and Rinaldi (1987).

In addition to the Cost and Time columns, we provide the values of p used in **GENI** and **GENIUS**.

Our results indicate that on GRID100 and GRID400, **GENI** (and thus **GENIUS**) finds an optimal solution. **GENIUS** also finds the optimum for K24. For the last two problems, **GENI** generates better solutions than all reported heuristics, except simulated annealing and tabu search. **GENIUS** is only surpassed by tabu search on a single problem (P532), with respect to solution cost, but appears to be much faster given the relative speeds of the two computers involved.

5. CONCLUSION

We have described a new insertion heuristic and a new postoptimization procedure for the TSP. Computational experiments carried out on both randomly generated problems and on problems described in the operations research literature confirm the relative efficiency of the two proposed methods. The **GENIUS** algorithm, which is derived from a combination of the two methods, compares advantageously with the best alternative heuristic approaches for the TSP, both in terms of solution quality and computation times. The proposed routines can be used in conjunction with other heuristics and can easily be extended to a number of VRPs and to other types of problems in which a minimum cost permutation of n objects must be determined. Finally, the idea of integrating local improvement steps within a constructive algorithm is of wide applicability in several areas of combinatorial optimization.

ACKNOWLEDGMENT

This work was partially supported by Canadian Natural Sciences and Engineering Research Council grants OGP0038816 and OGP0039682. The authors are grateful to Louis Salvail for his help with programming. Thanks are also due to the referees and to the associate editor for their valuable comments.

REFERENCES

Bartholdi, J. J., III, and L. K. Platzman. 1988. Heuristics Based on Spacefilling Curves for Combinatorial Problems in Euclidean Space. *Mgmt. Sci.* **34,** 291–305.

Bland, R. G., and D. F. Shallcross. 1989. Large Traveling Salesman Problems Arising From Experiments in X-Ray Crystallography: A Preliminary Report on Computation. *Opns. Res. Letts.* **8,** 125–128.

Cerny, V. 1985. Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm. *J. Optim. Theory Appl.* **45,** 41–51.

Chauny, F., A. Haurie, R. Loulou and E. Wagneur. 1987. Sequencing Punch Operations in an FMS: A Three-Dimensional Spacefilling Curve Approach. *INFOR* **25,** 26–45.

Crowder, H., and M. W. Padberg. 1980. Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality. *Mgmt. Sci.* **26,** 495–509.

Fiechter, C.-N. 1990. A Parallel Tabu Search Algorithm for Large Scale Traveling Salesman Problems. Working Paper 90/1 Department of Mathematics, École Polytechnique Fédérale de Lausanne, Switzerland.

Glover, F. 1977. Heuristic for Integer Programming Using Surrogate Constraints. *Dec. Sci.* **8,** 156–166.

Golden, B. L., and W. R. Stewart Jr. 1985. Empirical Analysis of Heuristics. In *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys (eds.). John Wiley, Chichester, U.K., 207–249.

Grötschel, M., M. Jünger and G. Reinelt. 1989. Via Minimization With Pin Preassignments and Layer Preference. *Z. Angew. Math. Mech.* **69,** 393–399.

Kirkpatrick, S., C. D. Gelatt Jr., and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* **220,** 671–680.

Krolak, P. D., W. Felts and G. Marble. 1971. A Man-Machine Approach Toward Solving the Traveling Salesman Problem. *Commun. ACM* **14,** 327–334.

Laporte, G., H. Mercure and Y. Nobert. 1986. An Exact Algorithm for the Asymmetrical Capacitated Vehicle Routing Problem. *Networks* **16,** 33–46.

Laporte, G., Y., Nobert and M. Desrochers. 1985. Optimal Routing Under Capacity and Distance Restrictions. *Opns. Res.* **33,** 1050–1073.

Lin, S. 1965. Computer Solutions of the Traveling Salesman Problem. *Bell Syst. Comput. J.* **44,** 2245–2269.

Lin, S., and B. W. Kernighan. 1973. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Opns. Res.* **21,** 498–516.

Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller. 1953. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.* **21,** 1087–1091.

Ong, H. L., and H. C. Huang. 1989. Asymptotic Expected Performance of Some TSP Heuristics. *Eur. J. Opnl. Res.* **43,** 231–238.

Or, I. 1976. Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking. Ph.D. Dissertation, Northwestern University, Evanston, Ill.

Padberg, M. W., and G. Rinaldi. 1987. Optimization of a 532-City Symmetric Traveling Salesman Problem by Branch and Cut. *Opns. Res. Letts.* **6,** 1–7.

PADBERG, M. W., AND G. RINALDI. 1990. Facet Identification for the Symmetric Traveling Salesman Problem. *Math. Program.* **47,** 219–257.

PREPARATA, F. P., AND M. I. SHAMOS. 1985. *Computational Geometry.* Springer-Verlag, New York.

REINELT, G. 1992. Fast Heuristics for Large Geometric Traveling Salesman Problems. *ORSA J. Comput.* **4,** 206–217.

ROSENKRANTZ, D. J., R. E. STEARNS AND P. M. LEWIS II. 1977. An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM J. Comput.* **6,** 563–581.

ROSSIER, Y., M. TROYON AND T. M. LIEBLING. 1986. Probabilistic Exchange Algorithms and the Euclidian Traveling Salesman Problem. *Opns. Res. Spektrum* **8,** 151–164.

SORIANO, P. 1989. Étude de Nouvelles Avenues de Recherche Proposées en Optimisation Combinatoire. Publication CRT-619, Centre de Recherche sur les Transports, University of Montreal, Montreal, Canada.

STEIGLITZ, K., AND P. WEINER. 1968. Some Improved Algorithms for Computer Solution of the Traveling Salesman Problem. In *Proceedings of the* 6th *Annual Allerton Conference on Circuit Theory,* R. T. Chien and T. N. Trick (eds.), New York, 814–821.

STEWART, W. R., JR. 1987. Accelerated Branch Exchange Heuristics for Symmetric Traveling Salesman Problems. *Networks* **17,** 423–437.

TROYON, M. 1988. Quelques Heuristiques et Résultats Asymptotiques pour Trois Problèmes d'Optimisation Combinatoire. Thèse No. 754, École Polytechnique Fédérale de Lausanne, Switzerland.