
Computational Optimization and Applications, 32, 231–257, 2005

Expanding Neighborhood GRASP for the Traveling Salesman Problem

YANNIS MARINAKIS marinakis@ergasya.tuc.gr
ATHANASIOS MIGDALAS sakis@verenike.ergasya.tuc.gr
Decision Support Systems Laboratory, Department of Production Engineering and Management, Technical University of Crete, 73100 Chania, Greece

PANOS M. PARDALOS pardalos@cao.ise.ufl.edu
Department of Industrial and Systems Engineering, University of Florida, USA

Received June 17, 2003; Revised May 28, 2004; Accepted October 11, 2004

Abstract. In this paper, we present the application of a modified version of the well known Greedy Randomized Adaptive Search Procedure (GRASP) to the TSP. The proposed GRASP algorithm has two phases: In the first phase the algorithm finds an initial solution of the problem and in the second phase a local search procedure is utilized for the improvement of the initial solution. The local search procedure employs two different local search strategies based on 2-opt and 3-opt methods. The algorithm was tested on numerous benchmark problems from TSPLIB. The results were very satisfactory and for the majority of the instances the results were equal to the best known solution. The algorithm is also compared to the algorithms presented and tested in the *DIMACS Implementation Challenge* that was organized by David Johnson [18].

Keywords: Traveling Salesman Problem, Greedy Randomized Adaptive Search Procedure, local search, Meta-Heuristics

1. Introduction

Consider a salesman who has to visit n cities. The Traveling Salesman Problem (TSP) asks for the shortest tour through all the cities such that no city is visited twice and the salesman returns at the end of the tour back to the starting city. We speak of a symmetric TSP, if for all pairs i, j the distance c_{ij} is equal to the distance c_{ji}. Otherwise, we speak of the asymetric traveling salesman problem. If the cities can be represented as points in the plain such that c_{ij} is the Euclidean distance between point i and point j, then the corresponding TSP is called the Euclidean TSP. Euclidean TSP obeys in particular the triangle inequality $c_{ij} \leq c_{ik} + c_{kj}$ for all i, j, k.

The Traveling Salesman Problem (TSP) is one of the most famous hard combinatorial optimization problems. Since 1950s many algorithms have been proposed, developed and tested for the solution of the problem. The TSP belongs to the class of NP-hard optimization problems [19]. This means that no polynomial time algorithm is known for its solution. Algorithms for solving the TSP may be divided into two classes, *exact algorithms* and *heuristic algorithms*.

The exact algorithms are guaranteed to find the optimal solution in exponential number of steps. The most effective exact algorithms are branch and cut algorithms [21] with which large TSP instances have been solved [2]. The problem with these algorithms is that they are quite complex and are very demanding of computer power [14]. For this reason it is very difficult to find optimal solution for the TSP, especially for problems with very large number of cities. Therefore, there is a great need for powerful heuristics that find good suboptimal solutions in reasonable amounts of computing time. These algorithms are usually very simple and have short running times. In the 1960s, 1970s and 1980s the attempts to solve the traveling salesman problem focused on tour construction methods and tour improvement methods. The proposed modified GRASP essentially combines these two approaches into a new algorithm. Construction methods build up a tour step by step. One of the simplest methods is the nearest neighbor in which, a salesman starts from an arbitrary city and goes to its nearest neighbor. Then, he proceeds from there in the same manner. There has been proposed a number of construction algorithms for the solution of the traveling salesman problem, including insertion heuristics, the Christofides algorithm, which is based on spanning trees in the underlying graph, and cost saving algorithms. The problem with construction heuristics is that although they are usually fast, they do not, in general, produce very good solutions. The improvement methods start with a tour and try to transform it into a shorter tour. The most known of these algorithms is the 2-opt heuristic, in which two edges are deleted and the open ends are connected in a different way in order to obtain another tour. In the general case, r edges in a feasible tour are exchanged for r edges not in that solution as long as the result remains a tour and the length of that tour is less than the length of the previous tour. The most powerful, and for many years the algorithm which performed best among all heuristic, is the Lin–Kernighan algorithm [24]. It decides dynamically at each iteration what the value of r (the number of edges to exchange) should be [14, 27].

In the last fifteen years, metaheuristics [1], such as simulated annealing, tabu search, genetic algorithms and neural networks, were introduced. These algorithms have the ability to find their way out of local optima. In simulated annealing [31], this is achieved by allowing the length of the tour even to increase with a certain probability. Gradually the probability allowing the objective function value to increase is lowered until no more transformations are possible. Tabu search [31] uses a different technique to get out of local optima. The algorithm keeps a list of forbidden transformations. In this way, it may be necessary to use a transformation that deteriorates the objective function value in the next step. Genetic algorithms [4, 31] mimic the evolution process in nature. Their basic operation is the mating of two tours in order to form a new tour. Moreover, they use algorithmic analogs to mutation and selection. Although there are few papers applying neural networks [26] algorithms to the TSP, the results obtained are not competitive with the mentioned heuristics [17].

In this paper, we use a modified version of the well known Greedy Randomized Adaptive Search Procedure (GRASP) for the solution of the TSP. It should be noted that in the *DIMACS Implementation Challenge* [12] which was organized by David Johnson, Fred Glover and Cesar Rego and presented in [18], no GRASP algorithm for the solution of the TSP is mentioned.

GRASP [7, 15, 16, 32] is an iterative two phase search which has gained considerable popularity in combinatorial optimization. Each iteration consists of two phases, a construction phase and a local search procedure. In the construction phase, a randomized greedy function is used to build up an initial solution. This randomized technique provides a feasible solution within each iteration. This solution is then exposed for improvement attempts in the local search phase. The final result is simply the best solution found over all iterations.

That is, in the first phase, a randomized greedy technique provides feasible solutions incorporating both greedy and random characteristics. This phase can be described as stepwise, adding one element at a time to the partial (incomplete) solution. The choice of the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidate in the list but not necessary the top candidate. The greedy algorithm is a simple one pass procedure for solving the traveling salesman problem. In the second phase, a local search is initialized from these points, and the final result is simply the best solution found over all searches (multi-start local search). Typically, the sequential approach is thought of as an iterative process. That is, in each iteration, phase one is used to generate a starting point, then the local search of phase two is applied, before proceeding to the next iteration. The GRASP algorithm may be described by the pseudo code below:

```
algorithm GRASP
do while stopping criteria not satisfied
    call GREEDY_RANDOM_SOLUTION(Solution)
    call LOCAL_SEARCH(Solution)
    if Solution is better than Best_Solution_Found then
        Best_Solution_Found ← Solution
    endif
enddo
return Best_Solution_Found
```

The neighborhood mapping used in the local search phase of the GRASP must also be defined. Of course, different problems require different construction and local search strategies. The advantage of the GRASP compared to other heuristics is that there are only two parameters to tune (the size of the candidate list and the number of GRASP iterations). Compared to tabu search, simulated annealing and genetic algorithms, GRASP appears to be competitive with respect to the quality of the produced solutions, the efficiency, and the fact that is easier to implement and tune. GRASP has been used extensively to solve difficult combinatorial optimization problems, including problems in scheduling, logic, location, assignment, transportation [7, 8, 16, 32] but also global optimization problems with combinatorial neighborhoods, such as the Capacitated Network Flow Problem (NCFP) [15, 16] and the Quadratic Assignment Problem (QAP) [28, 29]. Resende and Ribeiro [32]

present a recent survey of GRASP and Festa and Resende [8] present an extensive annotated bibliography on GRASP.

The new modified version of GRASP, implemented in this paper for the solution of the Traveling Salesman Problem (TSP), has a few differences from the original algorithm. The most significant of them concern the way the restricted candidate list (RCL) is constructed in the first phase and the strategy used for exploiting the neighborhood during the local search in the second phase. In most of the implementations of GRASP, some type of value based RCL construction scheme has been used. In such a scheme, an RCL parameter, α, determines the level of greediness or randomness in the construction. In our implementation the parameter α was not used and the best promising candidate edges are selected to create the RCL. Subsequently, one of them is chosen randomly to be the next candidate for inclusion to the tour. This type of RCL is called a cardinality based RCL construction scheme, and has not frequently used in the literature. In most algorithms for the solution of the Traveling Salesman Problem, as well as other difficult problems, a single local search algorithm has been used for the improvement of the initial solution. Here, it is proposed the use of a combination of two well known local search methods instead of a single local search. First, a restricted 2-opt method is applied to improve the solution of the first phase and, subsequently a restricted 3-opt method is tried in order to improve the 2-opt solution.

The structure of the paper is as follows. In the Section 2, an analytical description of the proposed modified Greedy Randomized Adaptive Search Procedure heuristic is presented. In the Section 3, the computational results are presented and, finally, concluding remarks and extensions are given in the last section.

2. Expanding neighborhood GRASP

In the previous section, the Greedy Randomized Adaptive Search Procedure was examined and its generic framework was presented. In this section, first a more detailed pseudocode of the proposed algorithm is presented and, then, the algorithm is explained step by step.

```
algorithm GRASP
do while stopping criteria not satisfied
    S = ∅ ! S is the current solution
    !E = number of arcs
    call Make_Queue(E)
    call Init_Set(parent, number of nodes)
    k = 0
    do while k < D ! D is the size of the RCL
        call Select_Min_From_List((i,j), E)
        call Delete_Min_From_List((i,j), E)
        Add l = (i, j) to RCL
        k = k + 1
    enddo
    m = 0
```

do while $m < n - 1$! where n is the number of nodes
  Select $l = (i, j)$ randomly from RCL
  root1 = `Collapsing_Find_Set`(i,parent)
  root2 = `Collapsing_Find_Set`(j,parent)
  if $root1 \neq root2$ **then**
    m = m + 1
    call `Merge_Subtours`((i, j), S)
    call `Union`(root1,root2,parent)
    Add the $(D + m)_{ith}$ edge to the RCL
  endif
enddo
call `Calc_Cost`(S, $cost\_S$)
! 2-opt
Given S ! where S is the current solution
$!E_1$ = number of arcs of the initial solution
call `Make_Queue`(E_1)
for $i = 1$ to n
  call `Select_Max_From_List`((i, j), E_1)
  !let (l, k) an edge of the current tour
  do for all possible (l, k)
  $S' = S \setminus (i, j) \setminus (l, k) \cup (i, l) \cup (j, k)$
  call `Calc_Cost`(S', $cost_{S'}$)
  if ($cost_{S'} < cost_S$) **then**
    Update the solution $S \leftarrow S'$
    $cost_S = cost_{S'}$
    call Delete_from_List(i, j)
    call Delete_from_List(l, k)
    call Add_to_List(i, l)
    call Add_to_List(j, k)
  endif
  enddo
endfor
! 3-opt
Given S ! where S is the current solution
$!E_1$ = number of arcs of the initial solution
call `Make_Queue`(E_1)
for $i = 1$ to n
  call `Select_Max_From_List`((i, j), E_1)
  !let (l,k) and (m,n) two edges of the current tour
  do for all possible (l, k)
  $S' = S \setminus (i, j) \setminus (l, k) \setminus (m, n) \cup (i, l) \cup (j, m) \cup (n, k)$
  call `Calc_Cost`(S', $cost_{S'}$)
  if ($cost_{S'} < cost_S$) **then**
    Update the solution $S \leftarrow S'$

```
                $cost_S = cost_{S'}$
                call Delete_from_List(i, j)
                call Delete_from_List(l, k)
                call Delete_from_List(m, n)
                call Add_to_List(i, l)
                call Add_to_List(j, m)
                call Add_to_List(n, k)
            endif
            enddo
        endfor
        if Solution S is better than Best_Solution_Found then
            Best_Solution_Found ← Solution S
        endif
    enddo ! GRASP iteration completed
    return Best_Solution_Found
```

In each iteration of GRASP a data structure for tour representation is needed. The choice of the data structure is a very significant part of the algorithm as it plays a critical role for the efficiency of the algorithm. We use the heap and the disjoint set data structures [33] in the construction phase. The heap is implemented as an integer array of pointers to an array of items. That is, we do not move items, we rather permute the pointers. It also uses an integer to keep the heap size. The heap is a complete binary tree represented as an array. The root is in position 1 of the binary tree. Any element I has its left child in position $2I$, and its right child in position $2I + 1$. Finally, parent is in the position $INT(I/2)$. Of course, the root has no parent. With the function *Make_Queue* an array is converted into a heap. The siftup/siftdown operations are used in adjusting the heap. The assumption is that the children of the root are already roots of subtrees that are in heap order. The functions *Select_Min_From_List* and *Delete_Min_From_List* are used for removing the root of the heap and in adjusting it.

The disjoint sets data structure implements tree representation of pairwise disjoint sets. The elements of the sets are integers. The representation of the sets is done using an integer array *parent*, where *parent*(I) is the parent of the element I in the tree where it belongs. If I is the root of the tree, then *parent*(I) is less or equal to 0. The function *Init_Set* initializes disjoint sets, where each set has one element (node), and the total number of sets is equal to number of nodes. The function *Collapsing_Find_Set* returns the root of the set in which the element belongs. At the same time, this function collapses the tree in order to reduce the longest path in the tree, giving, thus, a better complexity for a sequence of searches. Finally, the function *Union* joins two disjoint sets with root1 and root2, where root1 and root2 are values returned by function *Collapsing_Find_Set*, by making the smallest set a subtree of the other set.

The algorithmic representation of the complete graph was achieved by a list of arcs. In this data structure, it used 2 vectors for the arcs, one for the starting nodes and another for the ending nodes. It, also, used a third vector for the cost of each arc. This vector contains the Euclidean distances of all nodes.

In *stage one of the construction phase* it is created a list called Restricted Candidate List (RCL) that contains the best candidate edges for inclusion in the tour. Initially, a list of all the edges of a given graph G = (V,E) is created by ordering all the edges from the smallest to the largest cost using a heap data structure, as it was presented above. From this list, the first D, where D can vary from 30 to 150, are selected in order to form the Restricted Candidate List. This type of RCL is called cardinality based RCL. The candidate edge for inclusion in the tour is selected randomly from the RCL using a random number generator. Finally, the RCL is readjusted in every iteration by replacing the edge which has been included in the tour by another edge that does not belong to the RCL, namely the $(D + m)_{ith}$ edge where m is the number of the current iteration.

In the *stage two of the construction phase*, after the choice of an element for inclusion in the tour, it is used a modified version of Kruskal's algorithm, the nearest merger algorithm [22], for its insertion in partial tour. Kruskal's algorithm, when applied to a TSP of n nodes, constructs a sequence $S_1, \ldots, S_n$ such that each S_i is a set of $n - i + 1$ disjoint subtours covering all the nodes. It starts with n partial tours, each consisting of a single city, and successively merges the tours until a single tour containing all the cities is obtained. The procedure of merging is achieved with the function $Merge\_Subtours$. If the current number of tours exceeds one, the tours S, S' that should next be merged are chosen so that $min\{c_{ij} : i \in S \text{ and } j \in S'\}$ is as small as possible. The merging procedure have three possible alternatives (figure 1):

- If both S and S' consist of a single node then these two single node tours are merged in a tour.
- If S consists of a single city k, the merged tour is $TOUR(S', k)$. If $\{i, j\}$ is an edge of S' then $c_{ik} + c_{kj} - c_{ij}$ is minimized. The merged tour is then obtained by deleting $\{i, j\}$ and replacing them with $\{i, k\}$ and $\{j, k\}$. If S' consists of a single city k' then the merged tour is $TOUR(S, k')$. If $\{i, j\}$ is an edge of S then $c_{ik'} + c_{k'j} - c_{ij}$ is minimized. The merged tour is then obtained by deleting $\{i, j\}$ and replacing them with $\{i, k'\}$ and $\{j, k'\}$.
- If both S and S' contain at least two cities, let i, j, k and l be cities such that $\{i, j\}$ is an edge of S and $\{k, l\}$ is an edge of S' and $c_{ik} + c_{jl} - c_{ij} - c_{kl}$ is minimized. The merged tour is then obtained by deleting $\{i, j\}$ and $\{k, l\}$ and replacing them with $\{i, k\}$ and $\{j, l\}$.

After the completion of the construction phase, the initial solution of the algorithm is exposed for improvement. The improvement is achieved with the implementation of two different local search algorithms, namely 2-opt and 3-opt. A local search algorithm [25] is built around a neighborhood search procedure. A neighborhood N for the problem instance (S, g) can be defined as a mapping from S to its powerset: $N : S \rightarrow 2^S$. $N(s)$ is called the *neighborhood* of s. A solution x is called a local minimum of g with respect to the neighborhood N if $g(x) \leq g(y), \forall y \in N(x)$.

Given a feasible solution, the algorithm examines all the solutions that are closely related to this solution and finds a neighboring solution of lower cost, if one exists. Local search for the TSP is synonymous to k-opt moves. Using k-opt moves, neighboring solutions can be obtained by deleting k edges from the current tour and reconnected the resulting paths using k new edges. In the proposed algorithm, a combined neighborhood search is used.

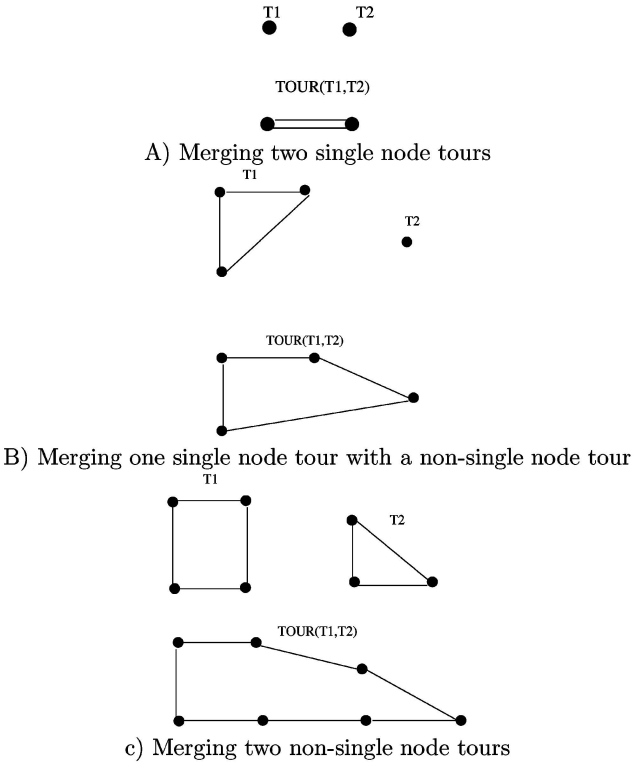


Figure 1. Example of modified Kruskall.

The idea of using a larger neighborhood to escape from a local minimum in a smaller one had been proposed initially by Garfinkel and Nemhauser [9] and recently by Hansen and Mladenovic [13]. First, the neighborhood function is defined as exchanging two edges of the current solution with two other edges. This procedure is known as 2-opt procedure and was introduced by Lin (1965) for the TSP [23]. Note that there is only one way to reconnect the paths. This algorithm does not use the classical approach of 2-opt but a restricted one. Initially the edges in the current tour are ordered from largest to smallest costs, using a heap data structure, and from this order, the edge with the largest cost is selected. The neighborhood to be examined is constructed from the end-nodes of this edge, but in a more sophisticated way than the one used in the classical 2-opt. Let i and j be the end-nodes of the edge and let l and k be the end-nodes of another candidate edge for deletion of the tour. The constructed neighborhood is then $S' = S \setminus (i, j) \setminus (l, k) \cup (i, l) \cup (j, k)$, where S is the initial solution, (i, j) and (l, k) are the candidate for deletion edges and (i, l) and (j, k) are the candidate for inclusion edges (figure 2). The cost of the candidate tour is, then, compared with the cost of the current tour and the first best tour found is used in order to replace the incumbent tour. The heap is then updated with the deletion of the edges (i, j) and (l, k) and the addition of the edges (i, l) and (j, k). The process is

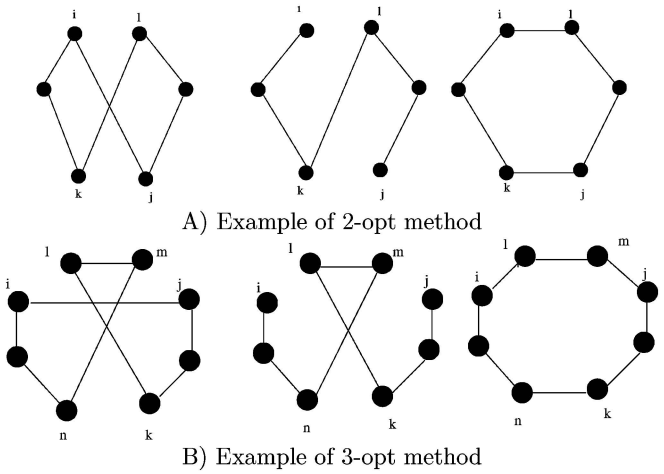


Figure 2. Examples of 2- and 3-opt.

repeated with the new tour using the edge with the largest cost from the updated heap. This procedure does not exclude a node of the current candidate for deletion edge to be the same as in a previous iteration. The process terminates when a number of candidate for deletion edges is examined. This number was selected to be equal to the number of nodes.

In the worst case, it can only be guaranteed that an improving move decreases the tour length by at least one unit. No polynomial worst case bound on the number of iterations to reach a local optimum can be given. Checking whether an improving 2–opt move exists takes $O(n^2)$ time.

Subsequently, the algorithm tries to improve the solution by expanding the neighborhood using a restricted 3-opt procedure. The 3-opt heuristic (figure 2) is quite similar the 2-opt. However, because it uses a larger neighborhood, it introduces more flexibility in modifying the current tour. The tour breaks into three parts instead of only two. There are eight ways to connect the resulting three paths in order to form a tour. There are $\binom{n}{3}$ ways to remove three edges from a tour. The overall best solution is kept. Thus, if 2-opt has been trapped in a local optimum, the algorithm has now the possibility to escape from it and find a better solution. Initially the edges in the current tour are ordered from largest to smallest costs, using a heap data structure, and from this order, the edge with the largest cost is selected. The neighborhood to be examined is constructed from the end-nodes of this edge, but in a more sophisticated way than the one used in the classical 3-opt. Let i and j be the end-nodes of the edge and let l, k and m, n be the end-nodes of two other candidate edges for deletion of the tour. The constructed neighborhood is then $S' = S\backslash(i, j)\backslash(l, k)\backslash(m, n) \cup (i, l) \cup (j, m) \cup (n, k)$, where S is the initial solution, (i, j), (l, k) and (m, n) are the candidate for deletion edges and (i, l), (j, m) and (n, k) are the candidate for inclusion edges (figure 2). The cost of the candidate tour is, then, compared with the cost of the current tour and the first best tour found is used in order to replace the incumbent tour. The heap is then updated with the deletion of the edges (i, j), (l, k) and

Table 1. Comparison between the solution of the modified GRASP and the best known solution.

Instance	Nodes	Best solution with GRASP	Best known solution	Quality (%)
Eil51	51	426	426	0
Berlin52	52	7542	7542	0
Eil76	76	538	538	0
Pr76	76	108159	108159	0
Rat99	99	1211	1211	0
KroA100	100	21282	21282	0
KroB100	100	22141	22141	0
KroC100	100	20749	20749	0
KroD100	100	21294	21294	0
KroE100	100	22068	22068	0
Rd100	100	7910	7910	0
Eil101	101	629	629	0
Lin105	105	14379	14379	0
Pr107	107	44303	44303	0
Pr124	124	59030	59030	0
Bier127	127	118326	118282	0.03
Ch130	130	6110	6110	0
Pr136	136	96772	96772	0
Pr144	144	58537	58537	0
Ch150	150	6528	6528	0
KroA150	150	26524	26524	0
Pr152	152	73682	73682	0
Rat195	195	2331	2323	0.34
D198	198	15788	15780	0.05
KroA200	200	29380	29368	0.04
KroB200	200	29482	29437	0.15
Ts225	225	126643	126643	0
Pr226	226	80414	80369	0.05
Gil262	262	2385	2378	0.29
Pr264	264	49135	49135	0
A280	280	2589	2579	0.38
Pr299	299	48235	48191	0.09
Rd400	400	15385	15281	0.68
Fl417	417	11895	11861	0.28
Pr439	439	107401	107217	0.17
Pcb442	442	50946	50778	0.33
D493	493	35253	35002	0.71
Rat575	575	6863	6773	1.32
P654	654	34707	34643	0.18
D657	657	49531	48912	1.26
Rat783	783	8897	8806	1.03
Pr1002	1002	262060	259045	1.16
Pcb1173	1173	57676	56892	1.37
D1291	1291	51616	50801	1.60
Rl1304	1304	255185	252948	0.88
Rl1323	1323	273115	270199	1.07
Fl1400	1400	20310	20127	0.90
Fl1577	1577	22427	22249	0.80
Rl1889	1889	319250	316536	0.85
D2103	2103	81312	80450	1.07
Pr2392	2392	386017	378032	2.11

Table 2. Improvement of the solution from the first phase by the 2-opt and 3-opt algorithms in the second phase.

Instance	Nodes	Kruskall	2-opt	3-opt	Average improvement Kruskall vs 2-opt (%)	Average improvement 2-opt vs 3-opt (%)
Eil51	51	469	438	426	6.60	2.73
Berlin52	52	11007	8545	7542	22.36	11.73
Eil76	76	770	582	538	24.41	7.56
Pr76	76	151873	119622	108159	21.23	9.58
Rat99	99	1728	1262	1211	26.96	4.04
KroA100	100	29764	23235	21282	21.93	8.40
KroB100	100	30287	24516	22141	19.05	9.68
KroC100	100	30770	23094	20749	24.95	10.15
KroD100	100	29764	22965	21294	23.08	7.27
KroE100	100	30369	24337	22068	19.86	9.32
Rd100	100	10912	8508	7910	22.03	7.02
Eil101	101	884	671	629	24.09	6.25
Lin105	105	21112	15764	14379	25.33	8.78
Pr107	107	56587	49836	44303	11.93	11.10
Pr124	124	84078	65369	59030	22.25	9.69
Bier127	127	154734	132037	118326	14.65	10.38
Ch130	130	8545	6852	6110	19.81	10.82
Pr136	136	143527	102003	96772	28.93	5.12
Pr144	144	87281	61162	58537	29.92	4.29
Ch150	150	9320	7045	6528	24.4	7.33
KroA150	150	37786	29397	26524	22.20	9.77
Pr152	152	97398	84183	73682	12.13	12.47
Rat195	195	3498	2546	2331	27.20	8.44
D198	198	19700	16383	15788	16.83	3.63
KroA200	200	40165	32315	29380	19.54	9.08
KroB200	200	41667	32622	29482	21.70	9.62
Ts225	225	202356	140775	126643	30.43	10.03
Pr226	226	112729	94360	80414	16.29	14.77
Gil262	262	3308	2616	2385	20.91	8.83
Pr264	264	71794	55517	49135	22.67	11.49
A280	280	3811	2807	2589	26.34	7.76
Pr299	299	68071	54251	48235	20.30	11.08
Rd400	400	21378	16618	15385	22.26	7.41
Fl417	417	15598	13172	11895	15.55	9.69
Pr439	439	153667	123147	107401	19.86	12.78
Pcb442	442	74886	56921	50941	23.98	10.50
D493	493	46089	38196	35253	17.12	7.70
Rat575	575	9587	7500	6863	21.76	8.49
P654	654	45809	38908	34707	15.06	9.39
D657	657	67962	55180	49531	18.80	10.23
Rat783	783	12623	9890	8897	21.65	10.04
Pr1002	1002	357617	287881	262060	19.50	8.96
Pcb1173	1173	85155	64569	57676	24.17	10.67
D1291	1291	78895	58253	51616	26.16	11.39
Rl1304	1304	388155	305154	255185	21.38	16.37
Rl1323	1323	418267	319161	273115	23.69	14.42
Fl1400	1400	26468	22320	20310	15.67	9.00
Fl1577	1577	33000	26508	22427	19.6	15.39
Rl1889	1889	478565	378945	319250	20.81	15.75
D2103	2103	142727	87563	81312	38.65	7.13
Pr2392	2392	552671	438540	386017	20.65	11.97

Table 3. Analytical presentation of solutions for four instances.

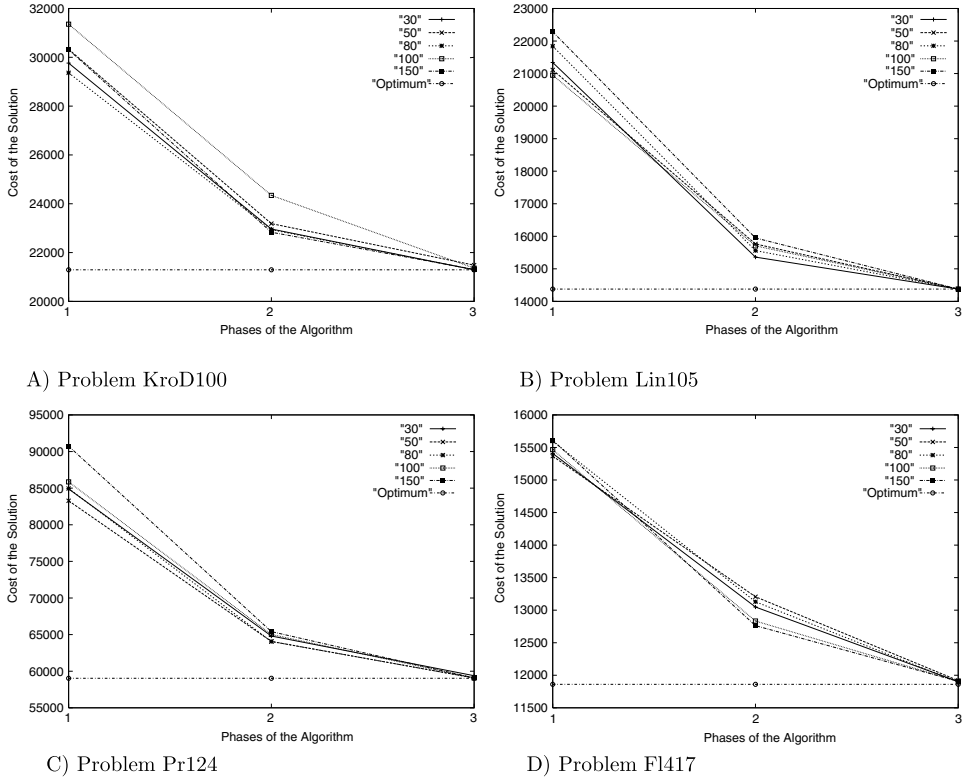
Instance	Phase	30	50	80	100	150	Optimum
KroD100							21294
	Kruskall	29764	30330	29364	31355	30311	
	2-opt	22965	23187	22937	24341	22834	
	3-opt	21294	21484	21294	21362	21309	
Lin105							14379
	Kruskall	21341	21112	21841	20950	22285	
	2-opt	15361	15764	15561	15706	15949	
	3-opt	14379	14379	14379	14379	14379	
Pr124							59030
	Kruskall	84895	83279	84942	85851	90729	
	2-opt	64804	64064	64061	65020	65407	
	3-opt	59385	59076	59076	59030	59076	
Fl417							11861
	Kruskall	15408	15366	15598	15466	15611	
	2-opt	13049	13206	13129	12833	12762	
	3-opt	11902	11918	11895	11908	11907	

(m, n) and the addition of the edges (i, l), (j, m) and (n, k). The process is repeated with the new tour using the edge with the largest cost from the updated heap. This procedure does not exclude a node of the current candidate for deletion edge to be the same as in a previous iteration. As before, the process terminates when a number of candidate for deletion edges is examined and this number was selected, again, to be equal with the number of nodes.

Each GRASP iteration concludes in a tour, resulting from the 3-opt procedure. The final result of the algorithm is simply the best solution over all iterations.

3. Computational results

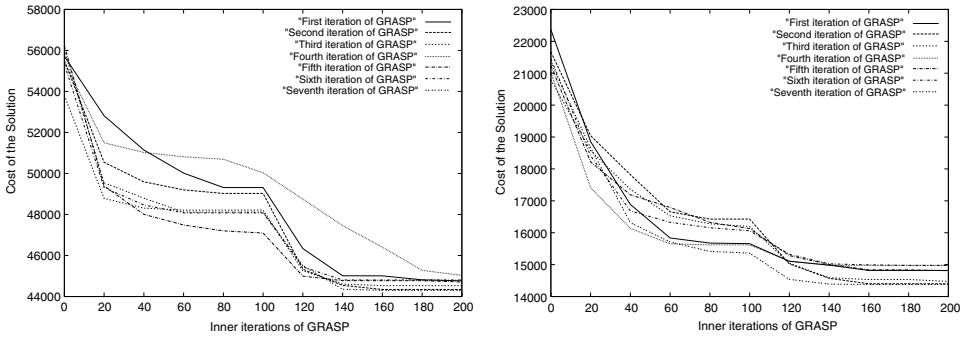
The modified GRASP was implemented in Fortran 90 and was compiled using the Linux VAST/ f90 compiler (f902f77 translator) on a Pentium III at 667 MHz, running Suse Linux 6.3. The test instances were taken from the TSPLIB (http://www.iwr.uni-heidelberg.de/groups/compt/software/TSPLIB95). The algorithm was tested on a set of 51 Euclidean sample problems with sizes ranging from 51 to 2392 nodes. Each instance is described by its TSPLIB name and size, e.g. the number 4 instance in Table 1 is named Pr76 with size equal to 76 nodes. In all tables, the first column shows the name of the instance while the second one shows the number of nodes. The length of RCL varies from 30 to 150. The number of iterations is between 10 and 100 depending on the number of nodes of each instance. For instances with a number of nodes over 1173 the algorithm was tested only for



A) Problem KroD100 B) Problem Lin105

C) Problem Pr124 D) Problem Fl417

Figure 3. Improvement of the solutions from the phases of the algorithm for different sizes of the RCL



A) Problem Pr107 B) Problem Lin105

Figure 4. Improvement of the solutions inside the phases of the algorithm

Table 4. Comparison of results for different sizes of the RCL.

Instance	Nodes	30	50	80	100	150	Best known solution (TSPLIB)
Eil51	51	426	426	426	428	426	426
Berlin52	52	7542	7542	7542	7542	7542	7542
Eil76	76	543	543	540	542	538	538
Pr76	76	108159	108159	108159	108159	108159	108159
Rat99	99	1219	1227	1211	1218	1211	1211
KroA100	100	21282	21282	21282	21282	21282	21282
KroB100	100	22141	22141	22141	22141	22141	22141
KroC100	100	20749	20852	20749	20749	20769	20749
KroD100	100	21294	21484	21294	21362	21309	21294
KroE100	100	22068	22068	22068	22068	22068	22068
Rd100	100	7910	7911	7911	7910	7911	7910
Eil101	101	634	632	637	629	631	629
Lin105	105	14379	14379	14379	14379	14379	14379
Pr107	107	44303	44347	44303	44303	44303	44303
Pr124	124	59385	59076	59076	59076	59030	59030
Bier127	127	120459	120162	118326	120338	119821	118282
Ch130	130	6172	6174	6170	6110	6131	6110
Pr136	136	98004	96772	97089	97537	97914	96772
Pr144	144	58537	58537	58537	58537	58537	58537
Ch150	150	6528	6555	6565	6570	6565	6528
KroA150	150	26704	26693	26618	26524	26585	26524
Pr152	152	73818	73662	73818	73818	73818	73682
Rat195	195	2359	2357	2363	2331	2351	2323
D198	198	15868	15883	15866	15788	15880	15780
KroA200	200	29798	29708	29637	29855	29380	29368
KroB200	200	29482	29654	29732	29878	29789	29437
Ts225	225	126643	126962	126643	126643	126643	126643
Pr226	226	81062	81000	80414	81005	81289	80369
Gil262	262	2424	2440	2385	2448	2429	2378
Pr264	264	49811	50124	49135	49923	49948	49135
A280	280	2649	2640	2632	2589	2645	2579
Pr299	299	49073	49016	48235	48911	48954	48191
Rd400	400	15613	15579	15385	15701	15559	15281
Fl417	417	11902	11918	11895	11908	11907	11861
Pr439	439	111606	111398	107401	109800	111300	107217
Pcb442	442	51490	51833	51815	50946	51497	50778
D493	493	36214	36166	36137	36481	35253	35002

10 iterations, except for the instances d1291 and fl1577 where the algorithm runs for 20 iterations.

In Table 1, the third column shows the best solution found by the modified GRASP, the fourth column shows the best solution taken from the TSPLIB, and the fifth column shows the quality of the solution produced by GRASP. The quality of the produced GRASP solutions is given in terms of the relative deviation from the best known solution, that is $p = \frac{100(c_{\text{grasp}} - c_{\text{opt}})}{c_{\text{opt}}}$, where c_{grasp} denotes the cost of the best solution found by GRASP, and c_{opt} is the cost of the best known solution.

It can be seen from Table 1 that the algorithm, in most instances with the number of nodes up to 264, has reached the best known solution. For the instances with the number of nodes between 280 and 2392 the quality of the solution is between 0.09 and 1.60%, except for the instance pr2392 where the quality is 2.11%. For the 51 instances for which the algorithm was tested, the best solution was found for twenty three of them, i.e. in 45% of all cases, for nineteen of them a solution was found with quality between 0.03 and 0.90%, i.e. 37.3%, for eight of them a solution was found with quality between 1.03 and 1.60%, i.e. 15.8%, and only one of them the quality of the produced solution is larger than 2.00%.

In Table 2, the third column shows the solution produced by Kruskall's algorithm, the fourth the solution by the 2-opt, the fifth the solution by the 3-opt, the sixth column indicates the average improvement obtained by the use of the 2-opt, while the last column shows the average improvement obtained by the use of the 3-opt. These results are from the GRASP iteration that gives the best solution for each instance. From Table 2, it is noted that in almost all instances the improvement obtained by applying 2-opt to the solution produced by Kruskall is about 20%. More precisely, the improvement varies from 6.60% in Eil51 to 38.65% in D2103. This difference is due to the quality of the initial solution, which for the Eil51 is 10.09%, while for the D2103 is 77.4%. The quality of the initial solution is computed as $p = \frac{100(c_{\text{Kruskall}} - c_{\text{opt}})}{c_{\text{opt}}}$, where c_{Kruskall} denotes the cost of the solution found by the first phase of the algorithm. For the instances where the improvement is about 20%, the quality of the initial solution varies between 30 and 40%. The improvement of the solution obtained by using 3-opt is about 10%. This improvement varies from 2.73% in Eil51 to 16.37% in Rl1304. Again, the improvement is based on the quality of the solution obtained by the 2-opt algorithm which is computed as $p = \frac{100(c_{2\text{opt}} - c_{\text{opt}})}{c_{\text{opt}}}$. The sequence of improvements can be seen from figure 4. The instances for which these results are given are presented in Table 3. The four chosen instances have different features with respect to their solutions. In the first, namely kroD100, the cost of the optimum solution is 21294 and the algorithm were able to find it for two different RCL sizes, namely 30 and 80. For the second instance, lin105, the algorithm were able to find the optimum for all five RCLs. In the third instance, pr124, the same local optimum was found for all the RCLs, except for the RCL with size equal to 30. The quality of this local optimum is 0.07%. For the fourth instance, fl417, the algorithm didn't find the optimum and produced different local optima for each size of the RCL. The quality of these local optima varies from 0.28%, for a list with size equal to 80, to 0.47% for a list with size equal to 50.

Table 4 presents the best results for different sizes of the RCL, for all instances. From this table, it is observed that, in most cases, a better solution was found when the size of RCL was 80. For all other RCL sizes the results are quite similar to each other and

Table 5. Relative running times of the phases of the GRASP.

Instance	Nodes	Kruskall (%)	2-opt (%)	3-opt (%)
Eil51	51	5	5	90
Berlin52	52	5	5	90
Eil76	76	4.9	1.6	93.5
Pr76	76	4.3	2.9	92.8
Rat99	99	3.6	2.2	94.2
KroA100	100	4.2	2.1	93.7
KroB100	100	3.4	1.4	95.2
KroC100	100	4.8	1.4	93.8
KroD100	100	4.7	1.3	94
KroE100	100	4.1	2.1	93.8
Rd100	100	4	1.3	94.7
Eil101	101	4.8	1.4	93.8
Lin105	105	4.8	1.8	93.4
Pr107	107	4.9	1.6	93.5
Pr124	124	4.4	1.5	94.1
Bier127	127	5.3	1.7	93
Ch130	130	3.1	1.6	95.3
Pr136	136	3.6	1.4	95
Pr144	144	3.2	1.4	95.4
Ch150	150	3.4	1.5	95.1
KroA150	150	3.2	1.6	95.2
Pr152	152	3.7	1.2	95.1
Rat195	195	2.5	1.2	96.3
D198	198	3.6	1.1	95.3
KroA200	200	2.3	1.2	96.5
KroB200	200	2.6	1.1	96.3
Ts225	225	2.4	1	96.6
Pr226	226	2.5	1.1	96.4
Gil262	262	1.8	0.9	97.3
Pr264	264	2.5	0.9	96.6
A280	280	3.3	0.8	95.9
Pr299	299	1.8	1	97.2
Rd400	400	2.1	0.6	97.3
Fl417	417	0.4	0.6	99
Pr439	439	1.2	0.6	98.2
Pcb442	442	1.4	0.5	98.1
D493	493	1.2	0.4	98.4
Rat575	575	1	0.4	98.6
P654	654	0.7	0.3	99
D657	657	0.3	0.4	99.3
Rat783	783	0.8	0.3	98.9
Pr1002	1002	0.2	0.3	99.5
Pcb1173	1173	0.4	0.2	99.4
D1291	1291	0.2	0.16	99.64
Rl1304	1304	0.17	0.2	99.63
Rl1323	1323	0.18	0.19	99.63
Fl1400	1400	0.14	0.12	99.74
Fl1577	1577	0.15	0.13	99.72
Rl1889	1889	0.23	0.13	99.64
D2103	2103	0.23	0.12	99.65
Pr2392	2392	0.10	0.10	99.8

quite close to the best solution. In figure 4 the improvement of the solution in each of the seven first iteration of the GRASP for two instances, Lin105 and Pr107, is presented. The *x*-axis shows the number of inner iterations that is, 0 corresponds to the first phase (Kruskall's algorithm) of GRASP, 1 - 99 correspond to the iterations of the 2-opt algorithm, while 101 - 200 correspond to the iterations of the 3-opt algorithm. For both instances the optimum solution was found after the fortieth iteration of the 3-opt algorithm.

In Table 5, the third, fourth and fifth columns show the relative times of the three phases with respect to the total time. From this table, it can be observed that when the numbers of nodes is less than 127 the relative time for Kruskall is around 4.5%, for 2-opt is around 2.2% and for 3-opt is around 93.3%. As the number of nodes increased the relative time for 3-opt is increasing, reaching 99.8% when the number of nodes is 2392. This is due to the fact that the full 3-opt version is used.

The results obtained by GRASP are also compared with two different set of results; those presented in [20], and those presented in the DIMACS Implementation Challenge which occurred in 2000/2001. The first one considers only instances with a number of nodes less than 1000. Table 6 compares the proposed GRASP to the results of this first set. In

Table 6. Comparison between GRASP and the approaches by Junger et al.

Instance	Nodes	Junger 2-opt	Junger 3-opt	Junger LK1	Junger LK2	Junger ILK	GRASP
Lin105	105	8.42	0.00	0.77	0.00	0.00	0.00
Pr107	107	3.74	2.05	1.53	0.81	0.00	0.00
Pr124	124	2.58	1.15	2.54	0.39	0.00	0.00
Pr136	136	10.71	6.14	0.55	0.72	0.38	0.00
Pr144	144	3.79	0.39	0.56	0.06	0.00	0.00
Pr152	152	2.93	1.85	0.00	0.19	0.00	0.00
Rat195	195	6.46	3.01	1.55	1.55	0.47	0.34
D198	198	3.85	6.12	0.63	1.51	0.16	0.05
Pr226	226	13.17	1.72	0.72	0.49	0.00	0.05
Gil262	262	10.62	3.07	1.18	2.44	0.55	0.29
Pr264	264	4.39	6.04	0.12	0.01	0.49	0.00
Pr299	299	10.46	4.37	1.55	1.36	0.15	0.09
Rd400	400	5.01	3.42	2.34	1.41	0.75	0.68
Pr439	439	6.52	3.61	2.73	2.68	0.38	0.17
Pcb442	442	8.74	3.01	1.41	1.94	0.90	0.33
D493	493	9.37	3.32	2.23	1.47	0.84	0.71
Rat575	575	7.93	4.46	2.48	1.68	1.03	1.32
P654	654	14.89	0.62	4.14	2.95	0.03	0.18
D657	657	7.57	3.52	3.10	1.65	0.74	1.26
Rat783	783	9.07	4.22	1.94	1.77	0.91	1.03

Table 7. Solution from all the algorithms used from the comparisons in the ten chosen instances.

Method	Pr1002	Pcb1173	d1291	rl1304	rl1323	fl1400	fl1577	rl1889	d2103	pr2392
Optimal	259045	56892	50801	252948	270199	20127	22249	316536	80450	378032
GRASP	262060	57676	51616	255185	273115	20310	22427	319250	81312	386017
Tour Construction Algorithms										
Strip	394289	73779		553358	546842			643133		556394
BW-Strip	394289	73779		395731	397495			518485		556394
Spacefil	364559	80189		413582	438287			517608		545731
FRP	425046	91670		480826	496323			589213		608752
Classic Tour Construction Algorithms										
B-Greedy	311124	67220	61979	301783	309811	24847	26327	371663	91290	454371
C-Greedy	309389	68135	61356	306481	313306	24175	25522	378846	88915	454439
Boruvka	302666	68350	61517	294245	315845	24617	26235	360741	91238	449044
Q-Boruvka	311188	65321	57697	300826	327940	26527	27598	383198	88187	456121
NN	312691	72969	63753	325331	345973	26771	27445	385967	92399	469774
CHCI	295940	67059		302853	323994			375611		443643
RI	293925	66310		302295	325854			374435		440108
FI	286683	65501		310246	326644			371709		427610
CW	288356	63404		280769	297899			352748		423945
CCA	284609	63668		273796	294950			354690		421011
HKChrist	276255	59901	53780	271186	287734	22223	23607	339415	83307	401404
Simple Local Search Algorithms (2-Opt, 3-Opt, etc.)										
2opt-J	273632	60438	55604	266203	280892	20904	24631	333680	83563	403579
2opt-B	274280	60422		270501	285347			343755		407396
2opt-C	299708	66338	57114	292767	311702	23326	25216	359136	88150	444207
2.5opt-B	273782	59492		267903	286058			336762		405106
3opt-J	267444	58650	52957	261382	275426	20795	23507	329249	82738	390673
3opt-B	269682	59176		262126	276243			330682		390709
H2	273280	59451	54310	269231	281864	21925	24260	341443	82601	400727
H3	269039	58824	53611	265604	279034	20830	24055	334823	81755	398906
H4	267594	59011	53328	266921	276877	20555	23983	330116	82088	394614
GENI	284517	61673		277605	286638			333759		411325
GENIUS	271919	58407		273138	282055			328470		394431
Lin-Kernighan Implementations and Variants										
CLK	266313	58110	53418	263065	284992	20974	24254	322868	81314	390510
ACRLK	265552	59781	52787	264124	279139	20755	24899	329179	81871	389600
LKHKCh	262323	57657	51708	256305	272701	20503	22400	319873	80835	382869
LK-J	263615	57758	52051	256116	273245	20362	22457	320364	81819	384124

(*Continue on next page.*)

Table 7. (*Continued*).

Method	Pr1002	Pcb1173	d1291	rl1304	rl1323	fl1400	fl1577	rl1889	d2103	pr2392
LK-N	262468	58012	52175	256613	273768	20367	22879	320359	82215	385073
LK-NYYY	263396	57504	53092	255987	272753	20307	22972	321149	81245	382005
SCE	262288	57700	51604	258038	274580	20382	22746	324059	81885	383622
ALK	259045	56893	50861	256202	271346	20262		317578		380051
H-LK	259441	56995	51267	254415	270513	20347	23487	318556	80620	378923
Repeated Local Search Algorithms										
CCLK	259952	57198	51828	255270	271074	20260	22873	318094	80610	379917
ACRCLK	259657	56968	51118	253361	270763	20129	22854	318296	80665	380878
ILK-J	259045	56897	50868	252948	270226	20127	22254	317134	80450	378226
I3opt	261298	57032	50912	252948	270484	20169	22274	318719	80944	382058
ILK-N	261014	57637	51009	254557	270774	20279	22266	317712	81347	380956
ILK-NYYY	259045	56892	50801	252948	270226	20127	22249	316562	80495	378051
M-LK	259734	57126	51295	255390	271468	20376	22327	320948	81253	379376
K-MLK	259045	56992	50886	252948	270199	20164	22263	316540	80487	378032
VNS-2H	261758	57961	51429	254159	272383	20404	23197	327304	81642	392948
VNS-3H	261913	57535	51812	256010	271736	20175	22494	323158	81419	386662
Algorithms that Use Chained LK as a Subroutine										
BSDP	260293	57054	51072	255370	271636	20260	23817	318032	81737	379987
TourMerge	259045	56892	50801	252948	270226	20127	22249	316562	80466	378032
Metaheuristics										
TS22	269765	59942	53196	271259	281905	21600	23096	332538	81657	143688
TS2DB	268743	59187	52782	264218	278973	20379	22848	326583	81360	391195
TSLKLK	261323	57290	51764	254542	272387	21416	24205	318265	81007	380486
TSLKDB	261469	57627	51149	255890	273018	20296	23226	317666	81034	381715
TSSCSC	260683	57546	50956	254698	272195	20720	24864	319959	80849	380066
TSSCDB	261277	57527	50983	256020	270877	20185	23285	318805	80665	380309

the third and fourth columns of this table the quality of the results of the 2-opt and the 3-opt algorithm are presented, respectively. The fifth and sixth column of the table present the results obtained by two versions of the Lin Kernighan heuristic as they are described in [20], while in the seventh column the results obtained by the Iterated Lin Kernighan are presented [20]. The Iterated Lin Kernighan is considered as the most cost effective adaptation for the solution of the Traveling Salesman Problem according to Johnson in [17]. The final column shows the quality of the results obtained by the proposed GRASP algorithm. For all instances the results from the proposed GRASP are better than those from the first four methods (including the two versions of the Lin Kernighan). As for the fifth method (ILK), the GRASP results are better in all cases except for five instances where the differences in the quality of the solutions are between 0.05 to 0.62%.

Table 8. Quality of each of the solutions and average quality from all the algorithms used from the comparisons in the ten chosen instances.

Method	Pr1002	Pcb1173	d1291	rl1304	rl1323	fl1400	fl1577	rl1889	d2103	pr2392	Average
GRASP	1.16	1.37	1.60	0.88	1.07	0.90	0.80	0.85	1.07	2.11	1.181
Tour Construction Algorithms											
Strip	52.21	29.68		118.76	102.38			103.18		47.18	75.565
BW-Strip	52.21	29.68		56.45	47.11			63.80		47.18	49.405
Spacefil	40.73	40.95		63.50	62.21			63.52		44.36	47.545
FRP	64.08	61.13		90.09	83.69			86.14		61.03	74.36
Classic Tour Construction Algorithms											
B-Greedy	20.10	18.15	22.00	19.31	14.66	23.45	18.33	17.42	13.47	20.19	18.708
C-Greedy	19.43	19.76	20.78	21.16	15.95	20.11	14.71	19.68	10.52	20.21	18.23
Boruvka	16.84	20.14	21.09	16.33	16.89	22.31	17.92	13.97	13.41	18.78	17.76
Q-Boruvka	20.13	14.82	13.57	18.93	21.37	31.80	24.04	21.06	9.62	20.66	19.6
NN	20.71	28.26	25.50	28.62	28.04	33.01	23.35	21.93	14.85	24.27	24.85
CHCI	14.24	17.87		19.73	19.91			18.66		17.36	17.96
RI	13.46	16.55		19.51	20.60			18.29		16.42	17.47
FI	10.67	15.13		22.65	20.89			17.43		13.11	16.64
CW	11.32	11.45		11.00	10.25			11.44		12.15	11.26
CCA	9.87	11.91		8.24	9.16			12.05		11.37	10.43
HKChrist	6.64	5.29	5.86	7.21	6.49	10.41	6.10	7.23	3.55	6.18	6.496
Simple Local Search Algorithms (2-Opt, 3-Opt, etc.)											
2opt-J	5.63	6.23	9.45	5.24	3.96	3.86	10.71	5.42	3.87	6.76	6.113
2opt-B	5.88	6.20		6.94	5.61			8.60		7.77	6.83
2opt-C	15.70	16.60	12.43	15.74	15.36	15.89	13.34	13.46	9.57	17.51	14.56
2.5opt-B	5.69	4.57		5.91	5.87			6.39		7.16	5.93
3opt-J	3.24	3.09	4.24	3.33	1.93	3.32	5.65	4.02	2.84	3.34	3.5
3opt-B	4.11	4.01		3.63	2.24			4.47		3.35	3.635
H2	5.50	4.50	6.91	6.44	4.32	8.93	9.04	7.87	2.67	6.00	6.218
H3	3.86	3.40	5.53	5.00	3.27	3.49	8.12	5.78	1.62	5.52	4.559
H4	3.30	3.72	4.97	5.52	2.47	2.13	7.79	4.29	2.04	4.39	4.062
GENI	9.83	8.40		9.75	6.08			5.44		8.81	8.05
GENIUS	4.97	2.66		7.98	4.39			3.77		4.34	4.685
Lin-Kernighan Implementations and Variants											
CLK	2.81	2.14	5.15	4.00	5.47	4.21	9.01	2.00	1.07	3.30	3.916
ACRLK	2.51	5.08	3.91	4.42	3.31	3.12	11.91	3.99	1.77	3.06	4.308
LKHKCh	1.27	1.34	1.79	1.33	0.93	1.87	0.68	1.05	0.48	1.28	1.202
LK-J	1.76	1.52	2.46	1.25	1.13	1.17	0.93	1.21	1.70	1.61	1.41
LK-N	1.32	1.97	2.70	1.45	1.32	1.19	2.83	1.21	2.19	1.86	1.804

(*Continue on next page.*)

Table 8. (*Continued*).

Method	Pr1002	Pcb1173	d1291	rl1304	rl1323	fl1400	fl1577	rl1889	d2103	pr2392	Average
LK-NYYY	1.68	1.08	4.51	1.20	0.95	0.89	3.25	1.46	0.99	1.05	1.706
SCE	1.25	1.42	1.58	2.01	1.62	1.27	2.23	2.38	1.78	1.48	1.702
ALK	0.00	0.00	0.12	1.29	0.42	0.67		0.33		0.53	0.42
H-LK	0.15	0.18	0.92	0.58	0.12	1.09	5.56	0.64	0.21	0.24	0.969
Repeated Local Search Algorithms											
CCLK	0.35	0.54	2.02	0.92	0.32	0.66	2.80	0.49	0.20	0.50	0.88
ACRCLK	0.24	0.13	0.62	0.16	0.21	0.01	2.72	0.56	0.27	0.75	0.567
ILK-J	0.00	0.01	0.13	0.00	0.01	0.00	0.02	0.19	0.00	0.05	0.041
I3opt	0.87	0.25	0.22	0.00	0.11	0.21	0.11	0.69	0.61	1.06	0.413
ILK-N	0.76	1.31	0.41	0.64	0.21	0.76	0.08	0.37	1.11	0.77	0.642
ILK-NYYY	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.06	0.01	0.009
M-LK	0.27	0.41	0.97	0.97	0.47	1.24	0.35	1.39	1.00	0.36	0.743
K-MLK	0.00	0.18	0.17	0.00	0.00	0.18	0.06	0.00	0.05	0.00	0.064
VNS-2H	1.05	1.88	1.24	0.48	0.81	1.38	4.26	3.40	1.48	3.95	1.993
VNS-3H	1.11	1.13	1.99	1.21	0.57	0.24	1.10	2.09	1.20	2.28	1.292
Algorithms that Use Chained LK as a Subroutine											
BSDP	0.48	0.28	0.53	0.96	0.53	0.66	7.05	0.47	1.60	0.52	1.308
TourMerge	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.02	0.00	0.004
Metaheuristics											
TS22	4.14	5.36	4.71	7.24	4.33	7.32	3.81	5.06	1.50	4.35	4.83
TS2DB	3.74	4.03	3.90	4.46	3.25	1.25	2.69	3.17	1.13	3.48	3.11
TSLKLK	0.88	0.70	1.90	0.63	0.81	6.40	8.79	0.55	0.69	0.65	2.2
TSLKDB	0.94	1.29	0.69	1.16	1.04	0.84	4.39	0.36	0.73	0.97	1.241
TSSCSC	0.63	1.15	0.31	0.69	0.74	2.95	11.75	1.08	0.50	0.54	2.034
TSSCDB	0.86	1.12	0.36	1.21	0.25	0.29	4.66	0.72	0.27	0.60	1.034

The proposed GRASP results are also compared with the results presented in the DIMACS Implementation Challenge (http://www.research. att.com/ dsj/chtsp/). This challenge is probably the most extensive examination made to date of heuristic algorithms in the field of TSP. Computational results for approximately 40 tour construction heuristics (divided in three categories, namely, Tour Construction Algorithms that Emphasize Speed over Quality, Classic Tour Construction Algorithms with quality $>15\%$ above the Held and Karp Bound and Classic Tour Construction Algorithms with quality $<15\%$ of the Held and Karp Bound) and for approximately 80 tour construction heuristics divided into 5 categories (namely, Simple Local Search Algorithms (2-Opt, 3-Opt, etc.), Lin-Kernighan Implementations and Variants, Repeated Local Search Algorithms, algorithms that Use Chained LK as a Subroutine and Metaheuristics (Tabu Search, Etc.)). In Tables 7 and 8, the results for ten instances, with number of nodes between 1002 and 2392, are presented.

Table 9. Ranking of the algorithms based on the average quality.

Rank	Method	Average
1	Tour Merging	0.004
2	ILK-NYYY	0.009
3	Iterated Lin - Kernighan by Johnson	0.041
4	Keld Helsgaun's Multi-Trial Variant on Lin-Kernighan	0.064
5	Iterated-3-Opt by Johnson	0.413
6	Augmented-LK	0.42
7	Applegate-Cook-Rohe Chained Lin–Kernighan	0.567
8	Iterated Lin–Kernighan by Neto	0.642
9	Multi-Level Lin-Kernighan Implementations	0.743
10	Concorde Chain Lin-Kernighan	0.88
11	Helsgaun Lin-Kernighan	0.969
12	Tabu-search-SC-DB	1.034
13	Expanding Neighborhood Search GRASP	1.181
14	Lin-Kernighan-with-HK-Christo-starts	1.202
15	Tabu-search-LK-DB	1.241
16	Variable-Neighborhood-Search-Using-3-Hyperopt	1.292
17	Balas-Simonetti-Dynamic-Programming-Heuristic	1.308
18	Johnson Lin–Kernighan	1.41
19	Stem–Cycle Ejection Chain	1.702
20	LK-NYYY	1.706
21	Neto Lin–Kernighan	1.804
22	Variable-Neighborhood-Search-Using-2-Hyperopt	1.993
23	Tabu-search-Stem - Cycle -SC	2.034
24	Tabu-search-LK-LK	2.2
25	Tabu-search-2opt-Double Bridge	3.11
26	3opt-J	3.5
27	3opt-B	3.635
28	Concorde-Lin-Kernighan	3.916
29	4-Hyperopt	4.062
30	Applegate Lin Kernighan	4.308
31	3-Hyperopt	4.559
32	GENIUS	4.685
33	Tabu-search-2opt-2opt	4.83
34	2.5opt-B	5.93
35	2opt-J	6.113
36	2-Hyperopt	6.218
37	Held Karp Christofides	6.496

(*Continue on next page.*)

Table 9. (*Continued.*)

Rank	Method	Average
38	2opt-B	6.83
39	GENI	8.05
40	CCA	10.43
41	Clarke–Wright	11.26
42	2opt-C	14.56
43	FI	16.64
44	RI	17.47
45	Boruvka	17.76
46	CHCI	17.96
47	C-Greedy	18.23
48	B-Greedy	18.708
49	Q-Boruvka	19.6
50	NN	24.85
51	Spacefilling	47.545
52	Best–Way Strip	49.405
53	FRP	74.36
54	Strip	75.565

The results are taken from the web page of the DIMACS Implementation Challenge (http://www.research.att.com/ dsj/ chtsp/). For the cases where the results are not given in the web page, the fields are left empty. In both tables, the first column gives abbreviated names for the algorithm. In Table 10 those abbreviations are explained and references to the literature are given. For completeness, we have included results for tour construction algorithms although these approaches are not competitive with methods which use a single or a more complicated local search phase. Since different runs of some algorithms may result in different solutions, the best results for each such approach are presented in Tables 7 and 8. On the other hand, the final column in Table 8 shows the average quality (over different runs) for these algorithms. This is used in order to rank the approaches in increasing order of average quality in Table 10, where the proposed GRASP is ranked thirteen among 54 algorithms.

As expected, the proposed GRASP ranks better than all tour construction heuristics. It also ranks better than all simple local search algorithms with exception of two Lin–Kernighan Implementations. It ranks better than all Variable Neighborhood Search implementations and it also ranks better than all metaheuristics but one Tabu implementation. The proposed GRASP gives slightly inferior results when compared to algorithms based on Iterated or Chained Lin Kernighan. It is known [18] that all metaheuristic–codes for STSP are dominated by 3-opt, Lin-Kernighan and Iterated Lin–Kernighan. However, as we mention in Section 4, it is possible to modify GRASP in order to take advantage of this fact.

Table 10. Abbrevated names used in the tables and explanation of them.

Abbrev	Short Description of heuristic and Reference Paper
Strip	Strip algorithm with initial sorting in x direction [18]
BW-Strip	Best-Way Strip algorithm, best of x- and y-direction strips, radix sort [18]
Spacefil	Spacefilling-Curve algorithm of Bartholdi and Platzman [18]
FRP	Fast-Recursive-Partitioning-Jon Bentley's kd-tree based implementation [5]
B-Greedy	Benchmark Greedy [18]
C-Greedy	Greedy:Concorde implementation [3]
Boruvka	Boruvka: Concorde's implementation [3]
Q-Boruvka	Quick-Boruvka—Concorde's implementation [3]
NN	Nearest-Neighbor—Concorde's implementation [3]
CHCI	Convex-Hull-Cheapest-Insertion (kd-tree based implementation) [5]
RI	Random-Insertion—Jon Bentley's kd-tree based implementation [5]
FI	Farthest-Insertion—Jon Bentley's kd-tree based implementation [5]
CW	Clarke-Wright-Savings Algorithm [6]
CCA	Golden-Stewart Convex Hull, Cheapest Insertion, Angle Selection Algorithm [18]
HKChrist	HeldKarp-Based-Christofides with Greedy Shortcuts [18]
2opt-J	2opt-JM-40-quadrant-neighbors with don't-look bits by Johnson [17]
2opt-B	2-Opt:Bentley kd-tree based implementation [5]
2opt-C	Concorde-2opt [3]
2.5opt-B	2.5-Opt:Bentley kd-tree based implementation [5]
3opt-J	3opt-JM-20-quadrant-neighbors[MIPS] with don't-look bits by Johnson [17]
3opt-B	3-Opt:Bentley kd-tree based implementation [5]
H2	2-Hyperopt [17]
H3	3-Hyperopt [17]
H4	4-Hyperopt [17]
GENI	GENI(p=10) [10]
GENIUS	GENIUS(p = 10) [10]
CLK	Concorde-Lin-Kernighan [3]
ACRLK	Applegate—Cook—Rohe Lin-Kernighan with 12 quadrant neighbors, don't-look bits [3]

(*Continue on next page.*)

Table 10. (*Continued.*)

Abbrev	Short Description of heuristic and Reference Paper
LKHKCh	Lin-Kernighan-with-HK-Christofides-starts [18]
LK-J	Lin-Kernighan-JM-40-quadrant-neighbors with don't-look bits by Johnson [17]
LK-N	Lin-Kernighan-Neto-20-quadrant-neighbors [27]
LK-NYYY	Nguyen-Yoshihara-Yamamori-Yasunaga Lin-Kernighan Variant [18]
SCE	Stem and Cycle Ejection Chain Algorithm [18]
ALK	Augmented LK [18]
H-LK	Helsgaun's Implementation of a powerful Lin-Kernighan Variant [14]
CCLK	Concorde-Chain Lin Kernighan [3]
ACRCLK	Applegate-Cook-Rohe Chained Lin-Kernighan [3]
ILK-J	Iterated Lin–Kernighan by Johnson [17]
I3opt	Iterated-3-Opt by Johnson [17]
ILK-N	ILK-Neto-(N/10)-iterations with cluster compensation, 40 quadrant neighbors [27]
ILK-NYYY	Nguyen-Yoshihara-Yamamori-Yasunaga Lin-Kernighan [18]
M-LK	Chris Walshaw's Multi-Level Lin-Kernighan Implementations [34]
K-MLK	Keld Helsgaun's Multi-Trial Variant on Lin-Kernighan [14]
VNS-2H	Variable-Neighborhood-Search-Using-2-Hyperopt [18]
VNS-3H	Variable-Neighborhood-Search-Using-3-Hyperopt [18]
BSDP	Balas-Simonetti-Dynamic-Programming-Heuristic [18]
TourMerge	Tour-merging (using a branch decomposition) of 10 Chained Lin-Kernighan tours [3]
TS22	Tabu-search-2opt-2opt [35]
TS2DB	Tabu-search-2opt-Double Bridge [35]
TSLKLK	Tabu-search-LK-LK [35]
TSLKDB	Tabu-search-LK-DB [35]
TSSCSC	Tabu-search-Stem - Cycle -SC [35]
TSSCDB	Tabu-search-SC-DB [35]

4. Conclusions

In this paper, a new algorithm for the solution of the traveling salesman problem was presented. The algorithm is a modification of the well known Greedy Randomized Adaptive Search Procedure. The results obtained using this algorithm were very satisfactory and in many cases (45% of the tested instances) equal to the best known solutions. For instances where the solution obtained is not equal to the optimal one, it was very close to it. Further improvement of the GRASP approach both with respect to solution quality and running times, would be possible to achieve by employing a different algorithm in the construction

phase, for instance one based on Prim's algorithm, and by better utilizing the concept of expanding neighborhood local search in the second phase. As it was shown in the comparisons of Table 9 the algorithm was ranked in the thirteen place between all the algorithms used in the DIMACS Implementation Challenge. The only approaches that show better performance than the proposed GRASP are those based on Iterated or Chained Lin Kernighan. However, GRASP is flexible enough to either incorporate the Iterated Lin Kernighan in its second phase or replace the 2- and 3- opt methods by such an approach. The investigation has also obviated the need for the development of satisfactory termination criteria for the GRASP approach. Work on these issues is under development.

References

1. E. Aarts and J.K. Lenstra, Local Search in Combinatorial Optimization, Wiley and Sons, 1997.
2. D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "On the solution of traveling salesman problem," Documenta Mathematica: Proc. Int. Cogr. Mathematica, vol. 3, pp. 645–656, 1998.
3. D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "Chained Lin-Kernighan for large traveling salesman problems," Informs Journal on Computing, (to appear).
4. R. Baralia, J.I. Hildago, and R. Perego, "A hybrid heuristic for the traveling salesman problem," IEEE Transactions on Evolutionary Computation, vol. 5, no. 6, pp. 1–41, 2001.
5. J.L. Bentley, "Fast algorithms for geometric traveling salesman problems," ORSA J. Computing, vol. 4, pp. 387–411, 1992.
6. G. Clarke and J.W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points", Operations Research, vol. 12, pp. 568–581, 1964.
7. T.A. Feo and M.G.C. Resende, "Greedy randomized adaptive search procedure," Journal of Global Optimization, vol. 6, pp. 109–133, 1995.
8. P. Festa and M.G.C. Resende, "GRASP: An annotated bibliography," in Essays and Surveys on Metaheuristics, C.C. Ribeiro and P. Hansen (Eds.), Kluwer Academic Publishers: Norwell, MA, 2001.
9. R. Garfinkel and G. Nemhauser, Integer Programming, Wiley and Sons, 1972.
10. M. Gendreau, A. Hertz, and G. Laporte, "New insertion and postoptimization procedures for the traveling salesman problem," Operations Research, vol. 40, pp. 1086–1094, 1992.
11. B.L. Golden and W.R. Stewart, "Empirical analysis of heuristics," in the Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, E.L. Lawer, J.K. Lenstra, A.H.G. Rinnoy Kan and D.B. Shmoys (Eds.), Wiley and Sons, 1985, pp. 207–249.
12. G. Gutin and A. Punnen, The Traveling Salesman Problem and Its Variations, Kluwer Academic Publishers Dordrecht, 2002.
13. P. Hansen and N. Mladenovic, "Variable neighborhood search: Principles and applications," European Journal of Operational Research, vol. 130, pp. 449–467, 2001.
14. K. Helsgaun, "An effective implementation of the lin-Kernighan traveling salesman heuristic," European Journal of Operational Research, vol. 126, pp. 106–130, 2000.
15. K. Holmqvist, A. Migdalas, and P.M. Pardalos, "Parallel continuous non-convex optimization," in Parallel Computing in Optimization, A. Migdalas, P.M. Pardalos, and S. Storøy (Eds.), Kluwer Academic Publishers, 1997, pp. 471–528.
16. K. Holmqvist, A. Migdalas, and P.M. Pardalos, "Parallelized heuristics for combinatorial search," in Parallel Computing in Optimization, A. Migdalas, P.M. Pardalos, and S. Storøy (Eds.), Kluwer Academic Publishers, 1997, pp. 269–294.
17. D.S. Johnson and L.A. McGeoch, "The traveling salesman problem: A case study," in Local Search in Combinatorial Optimization, E. Aarts and J.K. Lenstra (Eds.), Wiley and Sons, 1997, pp. 215–310.
18. D.S. Johnson and L.A. McGeoch, "Experimental Analysis of the STSP," in the Traveling Salesman Problem and Its Variations, G. Gutin and A. Punnen (Eds.), Kluwer Academic Publishers Dordrecht, 2002, pp. 369–444.

19. D.S. Johnson and C.H. Papadimitriou, "Computational complexity," in the Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, E.L. Lawer, J.K. Lenstra, A.H.D. Rinnoy Kan and D.B. Shmoys (Eds.), Wiley and Sons, 1985, pp. 37–85.
20. M. Junger, G. Reinhelt, and G. Rinaldi, "The traveling salesman problem," in Networks Models, Handbooks in OR and MS, M. Ball et al. (Eds.), Elsevier Science B.V, 1995, vol. 7, pp. 225–330.
21. G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," European Journal of Operational Research, vol. 59, pp. 231–247, 1992.
22. E.L. Lawer, J.K. Lenstra, A.H.G. Rinnoy Kan, and D.B. Shmoys, The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, Wiley and Sons, 1985.
23. S. Lin, "Computer solutions of the traveling salesman problem," Bell System Technical Journal, vol. 44, pp. 2245–2269, 1965.
24. S. Lin and B.W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," Operation Research, vol. 21, pp. 498–516, 1973.
25. Y. Marinakis and A. Migdalas, "Heuristic solutions of vehicle routing problems in supply chain management," in Combinatorial and Global Optimization, P.M. Pardalos, A. Migdalas, and R. Burkard (Eds.), World Scientific Publishing Co, 2002, pp. 205–236.
26. A. Modares, S. Somhom, and T. Enwaka, "A self - organizing neural network approach for multiple traveling salesman and vehicle routing problems," International Transactions in Operational Research, vol. 6, 1999, pp. 591–606.
27. D. Neto, "Efficient cluster compensation for Lin–Kernighan heuristics," PhD Thesis, Computer Science University of Toronto, 1999.
28. P.M. Pardalos, L. Pitsoulis, and M.G.C. Resende, "A parallel GRASP implementation for the quadratic assignment problem," in Solving Irregular Problems in Parallel—State of the Art, A. Ferreira and J. Rolim (Eds.), Kluwer Academic Publishers Dordrecht, 1995.
29. P.M. Pardalos, L. Pitsoulis, T. Mavridou, and M.G.C. Resende, "Parallel search for combinatorial optimization: Genetic algorithms, simulated annealing, tabu search and GRASP," in Solving Irregular Problems in Parallel - State of the Art, A. Ferreira and J. Rolim (Eds.), Kluwer Academic Publishers Dordrecht, 1995, pp. 317–331.
30. G. Reinhelt, The Traveling Salesman Problem, Computational solutions for TSP Applications, Springer-Verlag, 1994.
31. C. Rego and F. Glover, "Local search and metaheuristics," in the Traveling Salesman Problem and Its Variations, G. Gutin and A. Punnen (Eds.), Kluwer Academic Publishers Dordrecht, 2002, pp. 309–367.
32. M.G.C. Resende and C.C. Ribeiro, "Greedy randomized adaptive search procedures," in Handbooks of Metaheuristics, F. Glover and G.A. Kochenberger (Eds.), Kluwer Academic Publishers Dordrecht, 2003, pp. 219–249.
33. R. Tarjan, "Data structures and network algorithms," Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1983.
34. C. Walshaw, "A multilevel approach to the traveling salesman problem," Operations Research, (to appear).
35. M. Zachariasen and M. Dam, "Tabu search on the geometric traveling salesman problem," in Meta-heuristics: Theory and Applications, I.H. Osman and J.P. Kelly (Eds.), Kluwer Academic Publishers: Boston, 1996, pp. 571–587.