J Comb Optim (2009) 17: 134–156
DOI 10.1007/s10878-007-9104-2

Multiple phase neighborhood Search—GRASP based on Lagrangean relaxation, random backtracking Lin–Kernighan and path relinking for the TSP

Yannis Marinakis · Athanasios Migdalas · Panos M. Pardalos

Published online: 1 November 2007

Abstract In this paper, a new modified version of Greedy Randomized Adaptive Search Procedure (GRASP), called Multiple Phase Neighborhood Search—GRASP (MPNS-GRASP), is proposed for the solution of the Traveling Salesman Problem. In this method, some procedures have been included to the classical GRASP algorithm in order to improve its performance and to cope with the major disadvantage of GRASP which is that it does not have a stopping criterion that will prevent the algorithm from spending time in iterations that give minor, if any, improvement in the solution. Thus, in MPNS-GRASP a stopping criterion based on Lagrangean Relaxation and Subgradient Optimization is proposed. Also, a different way for expanding the neighborhood search is used based on a new strategy, the Circle Restricted Local Search Moves strategy. A new variant of the Lin-Kernighan algorithm, called Random Backtracking Lin-Kernighan that helps the algorithm to diversify the search in non-promising regions of the search space is used in the Expanding Neighborhood Search phase of the algorithm. Finally, a Path Relinking Strategy is used in order to explore trajectories between elite solutions. The proposed algorithm is tested on numerous benchmark problems from TSPLIB with very satisfactory results.

Keywords TSP · GRASP · ENS · Metaheuristics · Lagrangean relaxation and subgradient optimization

Y. Marinakis (✉) · A. Migdalas
Decision Support Systems Laboratory, Department of Production Engineering and Management, Technical University of Crete, 73100 Chania, Greece
e-mail: marinakis@ergasya.tuc.gr

A. Migdalas
e-mail: sakis@verenike.ergasya.tuc.gr

P.M. Pardalos
Department of Industrial and Systems Engineering, Center for Applied Optimization, University of Florida, Gainesville, FL 32611, USA
e-mail: pardalos@cao.ise.ufl.edu

1 Introduction

Greedy Randomized Adaptive Search Procedure (GRASP) (Feo and Resende 1995; Resende and Ribeiro 2003) is an iterative two phase search method which has gained considerable popularity in combinatorial optimization. Each iteration consists of two phases, a **construction phase** and a **local search phase**. 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. In the first phase, a **randomized greedy technique** provides feasible solutions incorporating both greedy and random characteristics. This phase can be described as a process which stepwise adds 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 during 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 candidates in the list but not necessarily 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 (c.f. multi-start local search).

The GRASP algorithm has been proven very efficient for the solution of the Traveling Salesman Problem (Marinakis et al. 2005a). Marinakis et al. (2005a) proposed an algorithm for the solution of the Traveling Salesman Problem that adds new features to the original GRASP algorithm.

In most GRASP implementations, some type of value based Restricted Candidate List (RCL) has been used. In such a scheme, an RCL parameter determines the level of greediness or randomness in the construction. In our implementation, this parameter is 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**. MPNS-GRASP introduces the flexibility of applying alternative greedy functions in each iteration instead of only one simple greedy function as in the classical approach. Moreover, a combination of greedy functions is also possible. The algorithm starts with one greedy function and if the results are not improving or if it leads to pathogenic circumstances (for example, in the case of TSP a pathogenic circumstance occurs when a number of very good arcs are left out of the solution), an alternative greedy function is used instead.

The utilization of a simple local search in the second phase of the classical algorithm limits the chances of obtaining better solutions. Thus, MPNS-GRASP uses instead, the Expanding Neighborhood Search which is a very flexible local search strategy.

Consequently, the MPNS-GRASP is a very flexible, powerful and adaptive algorithm. This happens because in MPNS-GRASP either the greedy function and/or the local search strategy are adaptively changed in case that for some specific problem

instance a combination of a greedy function with some local search strategy does not yield good results. Such a change is controlled by the use of two threshold parameters b and b_1, where b is the quality of the solution of the initial phase and b_1 is the quality of the solution of the second phase. The **quality** of the solution is given in terms of the relative deviation from the current Lagrangean Lower Bound. These parameters are examined every κ iterations.

Almost all GRASP implementations use a termination criterion based on the maximum allowed number of iterations. Consequently, the algorithm wastes time in iterations that only add small, if any, improvements. MPNS-GRASP, on the other hand, utilizes a termination criterion based on bounds obtained through Lagrangean Relaxation and Subgradient Optimization. That is, first, a lower bound (LBD) is calculated and an upper bound (UBD) is estimated using the techniques described in Marinakis et al. (2005b). Then, the parameter $e_{\text{bounds}} = \frac{UBD-LBD}{UBD}100\%$ is computed and if it is found to be less than a threshold value, the algorithm stops with the current solution of MPNS-GRASP.

The structure of the paper is as follows. In Sect. 2, a review of the methods used for the solution of the Traveling Salesman Problem is presented. In Sect. 3, an analytical description of the proposed Multiple Phase Neighborhood Search—Greedy Randomized Adaptive Search Procedure heuristic is presented. In Sect. 4, the computational results are presented and, finally, concluding remarks and extensions are given in the last section.

2 Traveling salesman problem

The Traveling Salesman Problem (TSP) is the problem of finding the shortest tour through all the cities that a salesman has to visit. The TSP is probably the most famous and extensively studied problem in the field of Combinatorial Optimization (Gutin and Punnen 2002; Lawer et al. 1985). The TSP belongs to the class of NP-hard optimization problems (Johnson and Papadimitriou 1985). 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 an exponential number of steps. The most effective exact algorithms are branch and cut algorithms (Laporte 1992) with which large TSP instances have been solved (Applegate et al. 2003). The problem with these algorithms is that they are quite complex and are very demanding of computer power (Helsgaum 2000). For this reason it is very difficult to find optimal solution for the TSP, especially for problems with a very large number of cities. Heuristic attempts to solve the Traveling Salesman Problem are focused on tour construction methods and tour improvement methods. Tour construction methods build up a solution step by step, while tour improvement methods start with an initial tour and then they try to transform it into a shortest tour (Johnson and McGeoch 1997). In the last twenty years, a breakthrough was obtained with the introduction of metaheuristics, such as simulated annealing, tabu search, genetic algorithms, ant colony optimization, particle swarm optimization, variable neighborhood search and neural networks. These algorithms have the possibility to find their way out of local optima

(Glover and Konchenberger 2003). A few years ago an implementation challenge was organized by Johnson (2002). In this challenge an effort was made to collect all possible algorithms that have been published in the area of Traveling Salesman Problem. Also, this challenge gave the opportunity to researchers to present, test and compare their methods with the benchmark methods in the area. The results of the challenge were presented in two papers (Johnson and McGeoch 2002; Johnson et al. 2002) and in the web page of the challenge http://www.research.att.com/~dsj/chtsp/.

3 Multiple phase neighborhood search GRASP (MPNS-GRASP)

3.1 General description of MPNS-GRASP

The Multiple Phase Neighborhood Search-GRASP (MPNS-GRASP) consists of an initialization phase, where the computations of a lower and an upper bound are conducted, and of three main phases. Each phase consists of a number of subphases. During initialization, a lower bound is first computed, and then a feasibility heuristic is applied in order to produce an initial feasible solution to the problem, that is the corresponding initial upper bound (Marinakis et al. 2005b). The process is repeated a number of times before the main phase of the algorithm begins in order to provide as good bounds as possible. Moreover, every κ ite rations (usually $\kappa = 10$) of the main phase, the lower and upper bounds are updated resulting into new values for the parameter e_{bounds}, and termination checks are performed based on these new values of e_{bounds}.

In the **first main phase**, initial feasible solutions are produced, in the **second main phase**, the Expanding Neighborhood Search is utilized in order to improve these solutions, and in the **third main phase**, a Path Relinking approach is used in order to explore trajectories between the solution and the current best solution. The first main phase of the algorithm consists of five subphases. In the *first subphase*, an array is converted into a heap using a heap data structure (Tarjan 1983). In the *second subphase*, the Restricted Candidate List (RCL) of the MPNS-GRASP is constructed. In the *third subphase*, the candidate elements for inclusion in the partial solution are selected randomly of the RCL, and the RCL is readjusted. In the *fourth subphase*, a step-wise construction algorithm is used for the construction of the initial feasible solution. In this subphase, the construction heuristic is selected automatically among a list of available heuristics. The choice is based on the threshold values b and b_1. Finally, in the *fifth* subphase after a number of iterations the quality of the best-so-far solution is checked and if it is greater than the threshold value b the greedy heuristic that is used in the construction phase is changed and the algorithm continues to create initial solutions with the use of the alternative greedy heuristic. The second main phase of the algorithm consists of the Expanding Neighborhood Search strategy (see Sect. 3.2.3) and the third main phase consists of the Path Relinking strategy (see Sect. 3.2.5). The steps of the MPNS-GRASP algorithm are given in detail below:

Initialization

1. Initial computation of lower and upper bounds, computation of e_{bounds}.
2. Select the set of the algorithms which will be used in the first main phase of the MPNS-GRASP.
3. Select the set of local search algorithms which will be used in the second main phase of the MPNS-GRASP.
4. Select the threshold values b and b_1 where:
   - b is the acceptance quality of the solution using a specified tour construction method in the first main phase of the algorithm after a prespecified number of iterations, and
   - b_1 is the acceptance quality of the solution using a specified local search strategy in the second main phase of the algorithm after a prespecified number of iterations.
5. Select the initial size of the RCL.

Main Algorithm

1. Set the number of iterations equal to zero.
2. **Do while** the stopping criteria are not satisfied (namely, e_{bounds} is greater than a threshold number or maximum number of iterations (1000) has not been reached):

   Main Phase 1

   1 Increase the iteration counter.
   2 Construct the RCL.
   3 Select randomly from the RCL the candidate element for inclusion to the partial solution.
   4 Call a greedy algorithm.
   5 If the number of iterations is equal to the prespecified number, check if the quality of the best solution for the first main phase is larger than the threshold b. In such a case choose another construction heuristic.

   Main Phase 2

   6 Call Expanding Neighborhood Search.

   Main Phase 3

   7 Call Path Relinking Process.

3. **Enddo**
4. Return the best solution.

3.2 Main phases of MPNS-GRASP

3.2.1 Data representation

In each iteration of the algorithm 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 (Tarjan 1983) 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.

3.2.2 Construction phase of MPNS-GRASP

Initially, the **Restricted Candidate List (RCL)** 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. From this list, the first D edges, where D can vary from 30 to 150, are selected in order to form the final Restricted Candidate List. This type of RCL is a **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 $(D + m)$th edge where m is the number of the current iteration. Once an element has been chosen for inclusion in the tour, a tour construction heuristic is applied in order to insert it in the partial tour.

The first tour construction heuristic used is a **modified version of Kruskal's algorithm** (Lawer et al. 1985). The modified version of Kruskal's algorithm, also called the nearest merger 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. If the current number of tours exceeds one, the tours S, S' that should be merged next, are chosen so that $\min\{c_{ij} : i \in S \text{ and } j \in S'$, where c_{ij} is the cost between i and $j\}$ is as small as possible. The merging procedure has three possible alternatives (Fig. 1):

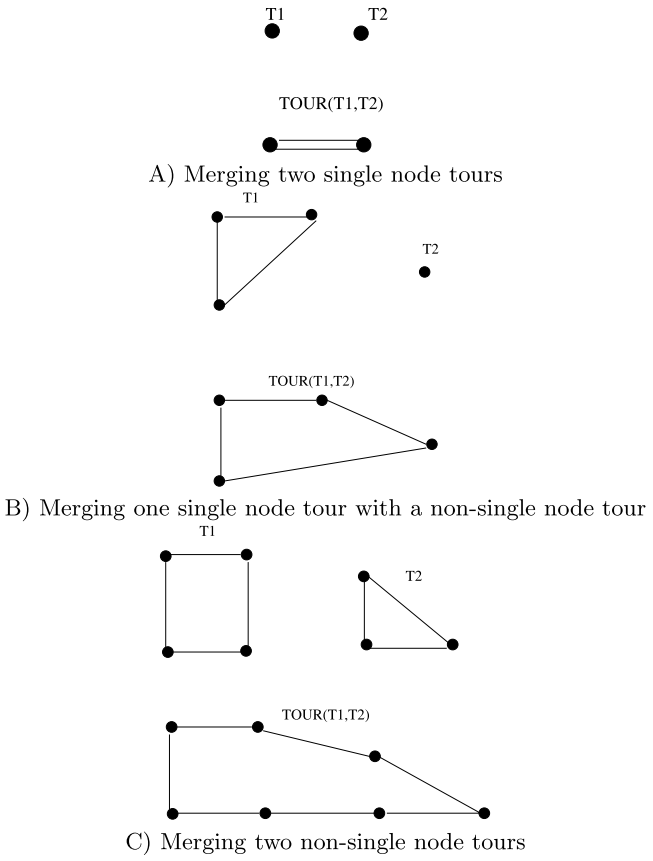

Fig. 1 Example of modified Kruskall

– 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 j_1, the merged tour is $TOUR(S', j_1)$. If (i, j) is an edge of S' then $c_{ij_1} + c_{j_1 j} - c_{ij}$ is minimized. The merged tour is then obtained by deleting (i, j) and replacing it with (i, j_1) and (j, j_1). If S' consists of a single city j'_1 then the merged tour is $TOUR(S, j'_1)$. If (i, j) is an edge of S then $c_{ij'_1} + c_{j'_1 j} - c_{ij}$ is minimized. The merged tour is then obtained by deleting (i, j) and replacing it with (i, j'_1) and (j, j'_1).
– If both S and S' contain at least two cities, let i, j, j_1 and i_1 be cities such that (i, j) is an edge of S and (j_1, i_1) is an edge of S', then $c_{ij_1} + c_{ji_1} - c_{ij} - c_{j_1 i_1}$ is minimized. The merged tour is then obtained by deleting (i, j) and (j_1, i_1) and replacing them with (i, j_1) and (j, i_1).

The second tour construction heuristic is the **pure greedy algorithm**. Initially, the degree (the number of edges that are connected to a node) of all the nodes is set equal to zero. Then, one edge at a time is selected randomly from the RCL. Each edge (i, j) is inserted in the partial tour taking into account that the degree of i and j should be

less than 2 and the new edge should not complete a tour with fewer than n vertices. In this algorithm, a number of paths are created which are finally joined to a single tour. For each candidate edge there are the following possibilities:

- If both end-nodes of the edge do not belong to any path (their degree is equal to 0), then they are joined together, a path is created and their degrees are increased by one.
- If one node belongs to a path and its degree is equal to 1 and the other node is a single node (its degree is equal to 0), then they are joined together and their degrees are increased by one.
- If both nodes belong to different paths and their degrees are equal to 1, then they are both in the beginning or in the end of their paths and the two paths are joined into a path. The degrees of the nodes are then increased by one.
- If both nodes have degree equal to 1 but they are in the same path, then there are two possibilities:
  1. If the path has less nodes than the number of nodes of the problem, then the edge is rejected. This happens because if the two nodes are joined then a tour with less nodes than the total number of nodes (a subtour) would be created.
  2. If the number of nodes in the path is equal to the number of nodes of the problem, then the two edges are joined producing thus a feasible tour.
- If one or both edges have degree equal to 2, then the edge is rejected and we proceed to the next edge.

3.2.3 Expanding neighborhood search

Expanding Neighborhood Search (ENS) has been proven very efficient for the solution of the Traveling Salesman Problem (Marinakis et al. 2005a). ENS is based on a method called **Circle Restricted Local Search Moves** and in addition, it has a number of local search phases.

In the **Circle Restricted Local Search Moves—CRLSM** strategy (Marinakis and Marinaki 2006; Marinakis et al. 2007), the computational time is decreased significantly compared to other heuristic and metaheuristic algorithms because all the edges that are not going to improve the solution are excluded from the search procedure. This happens by restricting the search into circles around the edges which are candidates for deletion.

In the following, a description of the Circle Restricted Local Search Moves Strategy for a **2-opt trial move** (Lin 1965) is presented. In this case, there are three possibilities based on the costs of the edges which are candidates for deletion and inclusion:

- If both new edges increase in cost, a 2-opt trial move can not reduce the cost of the tour (e.g., in Fig. 2 (Possibility A), for both new edges the costs $C2$ and $C4$ are greater than the costs $B2$ and A of both old edges).
- If one of the two new edges has cost greater than the sum of the costs of the two old edges, a 2-opt trial move again can not reduce the cost of the tour (e.g. in Fig. 2 (Possibility B) the cost of the new edge $C3$ is greater than the sum of the costs $A + B3$ of the old edges).

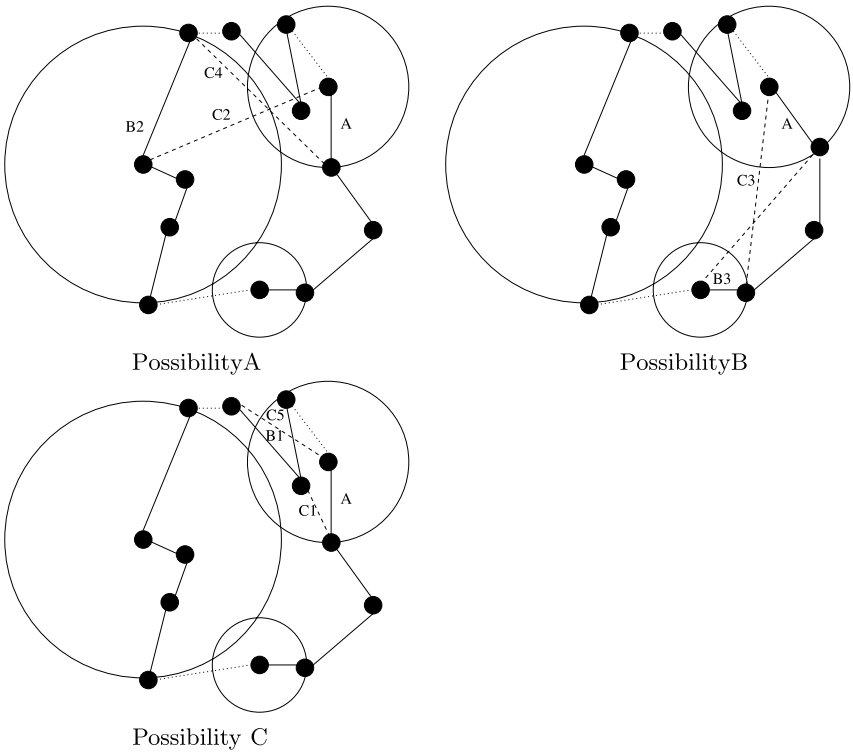

Fig. 2 Circle restricted local search moves strategy

– The only case for which a 2-opt trial move can reduce the cost of the tour is when at least one new edge has cost less than the cost of one of the two old edges (e.g., in Fig. 2 (Possibility C), the cost $C1$ of the new edge is less than the cost of the old edge A) and the other edge has cost less than the sum of the costs of the two old edges (e.g., $C5 < A + B1$ in Fig. 2 (Possibility C)).

Taking these observations into account, the Circle Restricted Local Search Moves strategy restricts the search to edges where one of their end-nodes is inside a circle with radius length at most equal to the sum of the costs (lengths) of the two edges which are candidates for deletion.

The ENS algorithm has the ability to change between different local search strategies. The idea of using a larger neighborhood to escape from a local minimum to a better one, had been proposed initially by Garfinkel and Nemhauser (1972) and recently by Hansen and Mladenovic (2001). Garfinkel and Nemhauser proposed a very simple way to use a larger neighborhood. In general, if with the use of one neighborhood a local optimum was found, then a larger neighborhood is used in an attempt to escape from the local optimum. On the other hand, Hansen and Mladenovic proposed a more systematical method to change between different neighborhoods, called Variable Neighborhood Search.

On the other hand, the Expanding Neighborhood Search Method starts with one prespecified length of the radius of the circle of the CRLSM strategy. Inside this circle a number of different local search strategies are applied until all the possible trial moves have been explored and the solution can not further be improved in this neighborhood. Subsequently, the length of the radius of the circle is increased, and again, the same procedure is repeated until the stopping criterion is activated. The main differences of ENS from the other two methods is the use of the Circle Restricted Local Search Move Strategy which restricts the search in circles around the edges which are candidates for deletion, the stopping criterion that is based on lower bounds to the optimal objective value, and finally, the more sophisticated way that the local search strategy can be changed inside the circles.

The idea of searching inside a radius of the circle of the neighborhood search, the Circle Restricted Local Search Moves Strategy, is the most innovative feature of the MPNS-GRASP. In the Expanding Neighborhood Search strategy, another innovative feature is the fact that the size of the neighborhood is **expanded** in each external iteration (Marinakis et al. 2007). Each different length of the neighborhood constitutes an external iteration. Initially, the size of the neighborhood, s, is defined based on the Circle Restricted Local Search Moves strategy, for example $s = A/2$, where A is the cost of one of the edges which are candidates for deletion. For the selected size of the neighborhood, a number of different local search strategies are applied until all the possible trial moves have been explored and the solution can not further be improved in this neighborhood. The local search strategies are changed based on two conditions, first if the current local search strategy finds a local optimum and second if the quality of the current solution remains greater than the threshold number b_1 for a number of internal iterations. Subsequently, the quality of the current solution is compared with the current Lagrangean lower bound. If the quality of the solution is less than the threshold number e_1 the algorithm stops, otherwise the neighborhood is expanded by increasing the length of the radius of the CRLSM strategy s by a percentage θ (e.g. $\theta = 10\%$), the Lagrangean lower bound is updated and the algorithm continues. When the length of the radius of the CRLSM strategy is equal to A, the length continues to increase until the length becomes equal to $A + B$, where B is the length of the other candidate for deletion edge. If the length of the radius of the CRLSM strategy is equal to $A + B$, and no stopping criterion has been already activated, then the algorithm terminates with the current solution. In Fig. 3, the Expanding Neighborhood Search Method is presented.

3.2.4 Local search

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 a combination of different local search algorithms, namely 2-opt, 2.5-opt, 3-opt and Lin–Kernighan (LK).

Local search for the TSP is synonymous to λ-opt moves. Using λ-opt moves, neighboring solutions can be obtained by deleting λ edges from the current tour and reconnecting the resulting paths using λ new edges. In the proposed algorithm, 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

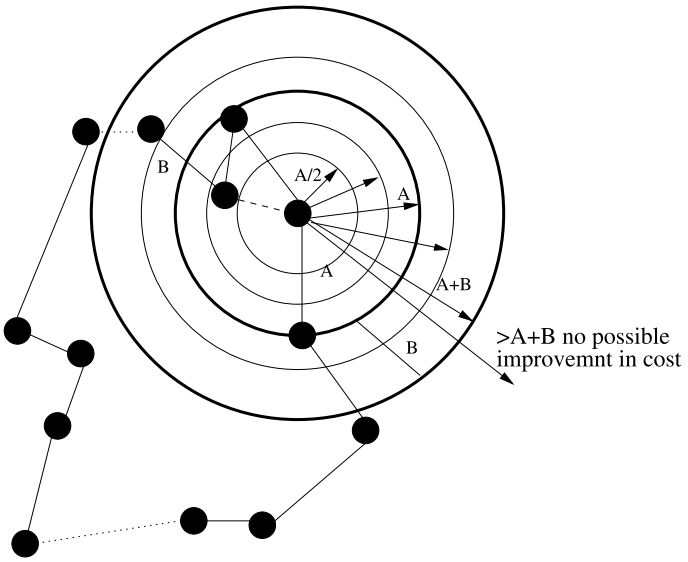

Fig. 3 Expanding neighborhood search method

introduced by Lin (1965) for the TSP. Note that there is only one way to reconnect the paths. Subsequently, the algorithm tries to improve the solution by expanding the neighborhood using a **modified 2.5-opt procedure**. This procedure is quite similar with 2-opt except that it has, additionally, a node insertion move. A selected node is inserted between 2 adjacent nodes in the new tour. The 3-opt heuristic is quite similar to 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.

The last local search strategy that is adopted is a modified version of Lin-Kernighan algorithm (LK), called **Random Backtracking Lin-Kernighan (RBLK)**. RBLK helps the algorithm to diversify the search in new regions of the search space. RBLK keeps the basic features of the initial LK as it was presented in (Lin and Kernighan 1973) but it uses a number of additional features in the backtracking process. The Random Backtracking Lin-Kernighan strategy is an algorithm for generating δ-opt moves where the value of δ is determined dynamically by performing a sequence of 2-opt moves. The pseudocode of the algorithm is the following:

RBLK Strategy
Given S ! S is the current solution
call `Make_Queue`(E_1) ! E_1 = number of arcs of the initial solution
do while No further improvement in the solution can be achieved
! PL = the number of levels of the process
! p_1 = the current level of the process, where $p_1 = 1, \ldots, PL$
$p_1 = 1$

```
    do while (p1 ≠ PL)
    if p1 = 1 then
        call Initial_Step
    else
        call Basic_Trial_Moves
    endif
    p1 = p1 + 1
    enddo
enddo
return S
```

With the function *Make_Queue* an array is converted into a heap, and the functions *Initial_Step* and *Basic_Trial_Moves* are explained below.

The new feature of this strategy is that in the backtracking phase instead of returning in the previous level, $p_1 - 1$, the algorithm chooses randomly in which of the previous levels to return. Let p_2 be the level that the algorithm will return to. If p_2 is close to the current level p_1, then the search for a better solution will continue in a neighborhood of solutions near to the current local optimum. However, if p_2 is far from p_1, the search will continue in a different region of the solution space (Fig. 4).

In our implementation, 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. Let i and j be the end-nodes of the edge and let (j_1, i_1), (i_2, j_2) and (i_3, j_3) be the end-nodes of three other edges that belong to the current tour. RBLK consists of a sequence of 2-opt trial moves. An exception is made for the first trial move, which can be either a 2-opt, a 3-opt or a 4-opt trial move.

- If the initial trial move is a 2-opt trial move the constructed neighborhood is $S' = S \setminus (i, j) \setminus (i_1, j_1) \cup (i, i_1) \cup (j, j_1)$, where S is the initial solution, (i, j) and (i_1, j_1) are the candidate for deletion edges and (i, i_1) and (j, j_1) are the candidate for inclusion edges. Then two different costs are calculated, the cost LM_1 that is equal to $c(i, j) - c(j, j_1)$ and the cost LT_1, that is equal to $c(i_1, j_1) - c(i, i_1)$. Finally, their summation $LD_1 = LM_1 + LT_1$ is calculated. If this sum is positive then the procedure continues with the deletion of the last added edge (i, i_1), else a 3-opt move is performed.
- If the initial move is a 3-opt move the procedure is analogous to the 2-opt case but the constructed neighborhood is $S' = S \setminus (i, j) \setminus (i_1, j_1) \setminus (i_2, j_2) \cup (i, i_2) \cup (j, j_1) \cup (j_2, i_1)$, where S is the initial solution, (i, j), (i_1, j_1) and (i_2, j_2) are the candidate for deletion edges and (i, i_2), (j, j_1) and (j_2, i_1) are the candidate for inclusion edges. Similarly, the costs LM_1 and LT_1 are calculated by $c(i, j) + c(i_1, j_1) - c(j, j_1) - c(j_2, i_1)$ and $c(i_2, j_2) - c(i, i_2)$, respectively. As previously, if the quantity $LD_1 = LM_1 + LT_1$ is positive, then the procedure continues with the deletion of the last added edge (i, i_2), else a 4-opt move is performed.
- If the initial move is a 4-opt move the constructed neighborhood is $S' = S \setminus (i, j) \setminus (i_1, j_1) \setminus (i_2, j_2) \setminus (i_3, j_3) \cup (i, i_3) \cup (j, j_1) \cup (j_2, i_1) \cup (i_2, j_3)$, where S is the initial solution, (i, j), (i_1, j_1), (i_3, j_3) and (i_2, j_2) are the candidate for deletion edges and (i, i_3), (j, j_1), (i_2, j_3) and (j_2, i_1) are the candidate for inclusion edges. The costs

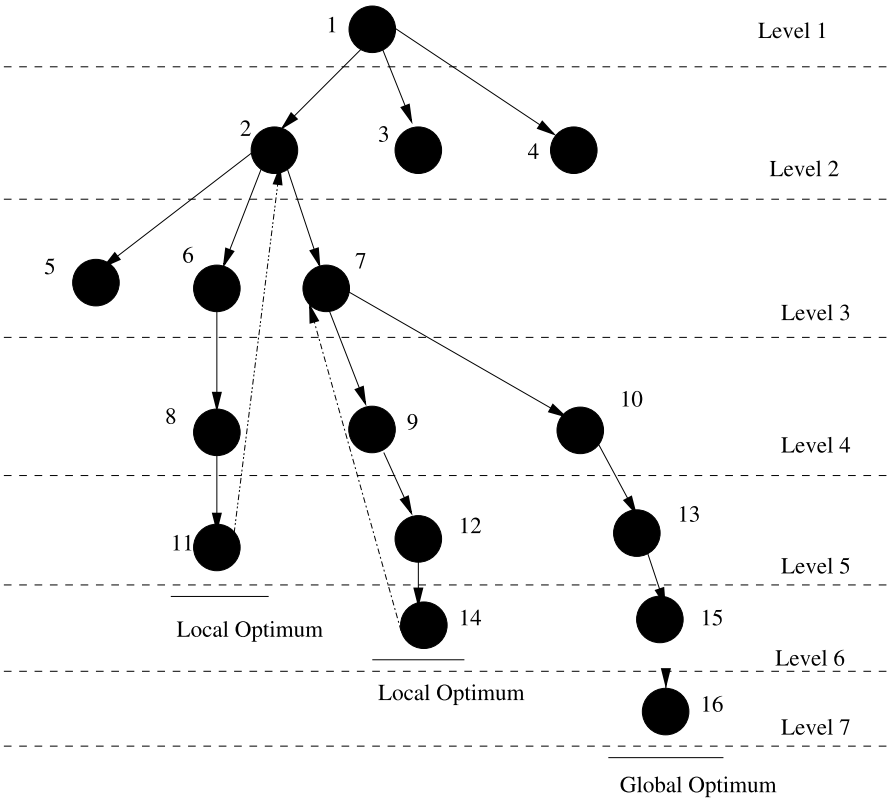

Fig. 4 Example of the random backtracking process of RBLK strategy

LM_1 and LT_1 are calculated by $c(i, j)+c(i_1, j_1)+c(i_2, j_2)-c(j, j_1)-c(j_2, i_1)-c(i_2, j_3)$ and $c(i_3, j_3) - c(i, i_3)$, respectively. As previously, if their summation $LD_1 = LM_1 + LT_1$ is positive then the procedure continues with the deletion of the last added edge (i, i_3), else the heap is updated and two new candidates edges are selected. When all the candidate edges have been selected without finding any positive value LD_1, the procedure is terminated.

In the following, a pseudocode for the initial trial move is given:

Initial_Step
! ***Initial 2-opt trial move***
call `Select_Max_From_List`$((i, j), E_1)$
Select (i_1, j_1) such that ! (i_1, j_1) = an edge of the current tour
$LM_{p_1} = c(i, j) - c(j, j_1) > 0$
$S' = S \setminus (i, j) \setminus (i_1, j_1) \cup (i, i_1) \cup (j, j_1)$
$LT_{p_1} = c(i_1, j_1) - c(i, i_1)$, $LD_{p_1} = LM_{p_1} + LT_{p_1}$
if $LD_{p_1} > 0$ **then**
call Re_Arrange_List(S)

Update the solution $S \longleftarrow S'$
Record the level p_1^* associated with the best $LD_{p_1^*}$ value
Keep the exchanges for using in backtracking
call Delete_from_List(i, i_1)
else ! ***alternative 3-opt trial move***
Select (i_1, j_1), (i_2, j_2) such that
$!(i_2, j_2) =$ an edge of the current tour
$LM_{p_1} = c(i, j) + c(i_1, j_1) - c(j, j_1) - c(j_2, i_1) > 0$
$S' = S \setminus (i, j) \setminus (i_1, j_1) \setminus (i_2, j_2) \cup (i, i_2) \cup (j, j_1) \cup (j_2, i_1)$
$LT_{p_1} = c(i_2, j_2) - c(i, i_2)$, $LD_{p_1} = LM_{p_1} + LT_{p_1}$
if $LD_{p_1} > 0$ **then**
call Re_Arrange_List(S)
Update the solution $S \longleftarrow S'$
Record the level p_1^* associated with the best $LD_{p_1^*}$ value
Keep the exchanges for using in backtracking
call Delete_from_List(i, i_2)
else ! ***alternative 4-opt trial move***
! $(i_3, j_3) =$ an edge of the current tour
Select (i_1, j_1), (i_2, j_2), (i_3, j_3) such that
$LM_{p_1} = c(i, j) + c(i_1, j_1) + c(i_2, j_2)$
$-c(j, j_1) - c(j_2, i_1) - c(i_2, j_3) > 0$
$S' = S \setminus (i, j) \setminus (i_1, j_1) \setminus (i_2, j_2) \setminus (i_3, j_3)$
$\cup (i, i_3) \cup (j, j_1) \cup (j_2, i_1) \cup (i_2, j_3)$
$LT_{p_1} = c(i_3, j_3) - c(i, i_3)$, $LD_{p_1} = LM_{p_1} + LT_{p_1}$
if $LD_{p_1} > 0$ **then**
call Re_Arrange_List(S)
Update the solution $S \longleftarrow S'$
Record the level p_1^* associated with the best $LD_{p_1^*}$ value
Keep the exchanges for using in backtracking
call Delete_from_List(i, i_3)
else
STOP
endif
endif
endif
return initial trial moves

The function *Select_Max_From_List* is used for selecting the root of the decreasing ordering heap. The function *Re_Arrange_List*(S) calls the functions *Delete_From_List* and *Add_to_List*, where the function *Delete_From_List* is used for removing the deleted edges of the heap and adjusting the list and the function *Add_to_List* is used for adding the selected edges in the heap and adjusting, again the heap.

For example, the function *Re_Arrange_List*(S) for the 2-opt strategy works as follows:

Re_Arrange_List(S)
call Delete_from_List(i, j)

call Delete_from_List(i_1, j_1)
call Add_to_List(i, i_1)
call Add_to_List(j, j_1)

If from any of these three trial moves a positive value of the LD_1 variant has been found, then the algorithm continues as follows. A sequence of 2-opt trial moves are generated in successive levels, where the 2-opt trial move of level p_1 drops one of the edges that has been added by the 2-opt trial move of the previous level $p_1 - 1$, where $p_1 = 2, \ldots, PL$ and PL is the depth of the RBLK procedure. Usually the value of PL is equal to the number of nodes in small problems while is set equal to 1000 in problems with the number of nodes greater than 1000. In each level the variants LM_{p_1}, LD_{p_1}, LT_{p_1} are calculated as follows:

- $LM_{p_1} = LM_{p_1-1} +$ (the difference between the edge that was dropped from the previous iteration with the edge that will replace it),
- $LT_{p_1} = LT_{p_1-1} +$ (the difference between the other two edges, the one that will be dropped from the tour and the one that it will be added in the tour), and
- $LD_{p_1} = LM_{p_1} + LT_{p_1}$.

In each level, the variant which is best so far LD_{p*}, is recorded and is compared with the variant LD_{p_1}. Also, in each level, the exchanges between the previous and the current level are recorded for using them in the backtracking process and in the replacement of the current solution with a new, better one when it is found. The process terminates when the number PL is achieved or the LD_{p_1} variant becomes negative.

Subsequently, a backtracking process is applied in which alternative trial moves are examined. The backtracking process is unlike those used in most implementations of the Lin-Kernighan algorithm. Except for the classic backtracking process that is applied in the first two levels, when a local optimum is found (meaning $LD_{p_1} < 0$), a backtracking process is applied in the next levels too. In this process, instead of returning to the previous level $p_1 - 1$, the algorithm chooses randomly in which of the previous p_2 levels it will return. Using this strategy there are two possibilities, first if the global optimum is near to the local optimum that the algorithm has been trapped then the solution can easily escape from the local optimum and move to a better solution, and second if p_2 is near to PL, then the process can easily search a different solution space that would have been remained unexplored if the classical backtracking process was used. A pseudocode for the basic moves of the RBLK is as follows.

Basic_Trial_Moves
! let (i_{p_1-1}, i) be the edge that is deleted in the previous iteration
! let (j_{p_1}, i_{p_1}) be an edge of the current tour
Select (j_{p_1}, i_{p_1}) such that
$LM_{p_1} = LM_{p_1-1} + c(i_{p_1-1}, i) - c(i_{p_1-1}, j_{p_1}) > 0$
$S' = S \setminus (i_{p_1-1}, i) \setminus (j_{p_1}, i_{p_1}) \cup (i, i_{p_1}) \cup (i_{p_1-1}, j_{p_1})$
$LT_{p_1} = c((j_{p_1}, i_{p_1})) - c(i, i_{p_1})$
$LD_{p_1} = LT_{p_1} + LM_{p_1}$
if $LD_{p_1} > 0$ **then**

call Re_Arrange_List(S)
Update the solution $S \longleftarrow S'$
Record the level p_1^* associated with the best $LD_{p_1^*}$ value
Keep the exchanges for using in backtracking
call Delete_from_List(i, i_{p_1})
else
!**Random Backtracking Process**
!Choose Randomly in which of the previous levels
!the algorithm will return
Call Random_number(p_2)
$p_2 = int(1 + p_2 * (20 - 1))$
Reverse all the exchanges that are accomplished between p_2
and the current level to p_1
Set $p_1 = p_2$
Continue the process with a different edge (j_{p_1}, i_{p_1})
endif
return basic moves

3.2.5 Path relinking

This approach generates new solutions by exploring trajectories that connect high-quality solutions—by starting from one of these solutions, called the *starting solution* and generating a path in the neighborhood space that leads towards the other solution, called the *target solution* (Glover et al. 2003). The roles of starting and target solutions can be interchangeable. In the first one, the worst among the two solutions plays the role of the starting solution and the other plays the role of the target solution. In the second one, the roles are exchanged. There is the possibility the two paths will be simultaneously explored. In the MPNS-GRASP, a path relinking strategy is applied where the current solution plays the role of the starting solution and the best solution of the whole population of solutions plays the role of the target solution. The trajectories between the two solutions are explored by simple swapping of two nodes of the starting solution until the starting solution becomes equal to the target solution. If in some step of the path relinking strategy a new best solution is found then the current best solution is replaced with the new one and the algorithm continues.

4 Computational results

The MPNS-GRASP was implemented in Fortran 90 and was compiled using the Lahey f95 compiler on a Centrino Mobile Intel Pentium M750 at 1.86 GHz, running Suse Linux 9.1. 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 74 Euclidean sample problems with sizes ranging from 51 to 85 900 nodes. Each instance is described by its TSPLIB name and size, e.g. in Table 1 the instance named Pr76 has size equal to 76 nodes. The parameters of the proposed algorithm were selected after thorough testing. A number of different alternative values were tested

Table 1 Parameter values

Parameter	Value
Number of iterations	1000
e_1	3% above the lower bound
b	15% above the lower bound
b_1	8% above the lower bound
κ	10
Size of RCL	30 to 150

and the ones selected are those that gave the best computational results concerning both the quality of the solution and the computational time needed to achieve this solution. Thus, the selected parameters are given in Table 1.

In this section, in addition to the results of the MPNS-GRASP, the lower bounds of the Lagrangean Relaxation and Subgradient Optimization are presented. The first main phase of the MPNS-GRASP algorithm begins with the Kruskal algorithm. During the iterations of the algorithm in the first main phase the pure greedy algorithm is also used while in the second main phase, the Expanding Neighborhood Search, 2-opt, 2.5-opt, 3-opt and RBLK are used.

In Table 2, the first column shows the name of the instance (the numbers in the names denote the nodes of each instance), the second column shows the best lower bound produced by Lagrangean Relaxation and Subgradient Optimization, the third and fourth columns present the quality of the lower bounds based on the optimal solution and the solution from the MPNS-GRASP with path relinking, respectively. The quality of the produced lower bounds is very good and in a very large number of instances (in 55 out of 74) is less than 3% of the optimum. The fifth column presents the solution of the MPNS-GRASP algorithm without using the path relinking strategy while the sixth column presents the results with the path relinking strategy. The seventh column presents the optimal solutions for each instance taken from the TSPLIB, the eighth column shows the **quality** of the solution produced by the MPNS-GRASP algorithm with path relinking, and finally the last column gives the computational time (in second) needed to find the solution. The quality of the produced solutions is given in terms of the relative deviation from the optimum, that is $\omega = \frac{100(c_{\text{MPNS-GRASP}} - c_{\text{opt}})}{c_{\text{opt}}}$, where $c_{\text{MPNS-GRASP}}$ denotes the cost of the optimum found by MPNS-GRASP with path relinking, and c_{opt} is the cost of the optimal solution.

As it was mentioned in Sect. 3.1, one of the stopping criteria used is that the algorithm stops when the quality of the solution becomes smaller from a threshold value. The threshold value is set equal to 2% of the quality of the lower bound. The use of this termination criterion is necessary because by its use the algorithm will be prevented from wasting time in iterations that only add small, if any, improvements to the solution. In forty three instances, the value of the lower bound is smaller than the threshold value. In the remaining 31 instances, when a maximum number of iterations is reached, the algorithm terminates. The benefit of the existence of an additional stopping criterion besides the maximum number of iterations is that the algorithm is more efficient and it consumes less computational time. In Fig. 5, the quality of

Table 2 Results of the MPNS-GRASP

Instance	LBD	Qual. LBD based on the Opt. (%)	Qual. LBD based on MPNS (%)	MPNS-GRASP without Path Relinking	MPNS-GRASP with Path Relinking	Optimum	Qual. (%)	CPU (s)
Eil51	396	7.04	7.04	426	426	426	0	0.05
Berlin52	7542	0.00	0.00	7542	7542	7542	0	0.08
Eil76	494	8.18	8.18	538	538	538	0	0.15
Pr76	105090	2.84	2.84	108159	108159	108159	0	0.15
Rat99	1157	4.46	4.46	1211	1211	1211	0	0.15
KroA100	20901	1.79	1.79	21282	21282	21282	0	0.16
KroB100	21729	1.86	1.86	22141	22141	22141	0	0.16
KroC100	20432	1.53	1.53	20749	20749	20749	0	0.15
KroD100	21026	1.26	1.26	21294	21294	21294	0	0.16
KroE100	21740	1.49	1.49	22068	22068	22068	0	0.16
Rd100	7844	0.83	0.83	7910	7910	7910	0	0.15
Eil101	568	9.70	9.70	629	629	629	0	0.25
Lin105	14282	0.67	0.67	14379	14379	14379	0	0.26
Pr107	43845	1.03	1.03	44303	44303	44303	0	0.26
Pr124	56985	3.46	3.46	59030	59030	59030	0	0.38
Bier127	117295	0.83	0.83	118282	118282	118282	0	0.49
Ch130	6026	1.37	1.37	6110	6110	6110	0	0.69
Pr136	94578	2.27	2.27	96772	96772	96772	0	0.89
Pr144	56134	4.11	4.11	58537	58537	58537	0	1.05
Ch150	6404	1.90	1.90	6528	6528	6528	0	1.12
KroA150	26152	1.40	1.40	26524	26524	26524	0	1.28
Pr152	72444	1.68	1.68	73682	73682	73682	0	1.34
Rat195	2204	5.12	5.12	2323	2323	2323	0	2.43
D198	15567	1.35	1.35	15780	15780	15780	0	2.72
KroA200	29001	1.25	1.25	29368	29368	29368	0	2.58
KroB200	29008	1.46	1.46	29437	29437	29437	0	2.51
Ts225	115504	8.80	8.80	126643	126643	126643	0	2.6
Pr226	77321	3.79	3.79	80369	80369	80369	0	3.61
Gil262	2227	6.35	6.35	2378	2378	2378	0	3.78
Pr264	45324	7.76	7.76	49135	49135	49135	0	4.16
A280	2454	4.85	4.85	2579	2579	2579	0	3.5
Pr299	47060	2.35	2.35	48235	48191	48191	0	6.8
Rd400	14963	2.08	2.08	15385	15281	15281	0	9.8
Fl417	10785	9.07	9.07	11895	11861	11861	0	11.2
Pr439	103196	3.75	3.75	107401	107217	107217	0	12.1
Pcb442	50247	1.05	1.05	50946	50778	50778	0	11.8
D493	34456	1.56	1.56	35225	35002	35002	0	15.7

Table 2 (*continued*)

Instance	LBD	Qual. LBD based on the Opt. (%)	Qual. LBD based on MPNS (%)	MPNS-GRASP without Path Relinking	MPNS-GRASP with Path Relinking	Optimum	Qual. (%)	CPU (s)
Rat575	6508	3.91	3.91	6789	6773	6773	0	16.12
P654	31780	8.26	8.26	34686	34643	34643	0	19.87
D657	48205	1.45	1.45	49176	48912	48912	0	21.76
Rat783	8415	4.44	4.44	8854	8806	8806	0	34.45
dsj1000	18537889	0.65	0.66	18712324	18661212	18659688	0.0081	35.12
Pr1002	256552	0.96	0.96	259090	259045	259045	0	38.79
u1060	222648	0.65	0.65	225087	224094	224094	0	39.28
vm1084	236156	1.31	1.31	240101	239297	239297	0	41.01
Pcb1173	56228	1.17	1.17	56915	56892	56892	0	41.25
D1291	50108	1.36	1.36	50915	50801	50801	0	45.37
Rl1304	248777	1.65	1.65	253950	252948	252948	0	58.98
Rl1323	264028	2.28	2.28	271252	270199	270199	0	63.25
nrw1379	56288	0.62	0.62	56824	56638	56638	0	66.89
Fl1400	19389	3.67	3.67	20235	20127	20127	0	89.98
u1432	151289	0.64	0.64	153284	152270	152270	0	88.74
Fl1577	21464	3.53	3.53	22321	22249	22249	0	102.34
d1655	61281	1.36	1.41	62543	62157	62128	0.0466	105.58
vm1748	331568	1.48	1.54	338654	336754	336556	0.0588	111.23
u1817	56579	1.09	1.09	57305	57201	57201	0	118.17
Rl1889	309980	2.07	2.07	317278	316536	316536	0	112.56
D2103	78980	1.83	1.83	80538	80450	80450	0	136.78
u2152	63348	1.41	1.46	64560	64288	64253	0.0544	148.24
u2319	233588	0.29	0.29	235609	234256	234256	0	149.38
Pr2392	368653	2.48	2.48	379120	378032	378032	0	147.34
pcb3038	136424	0.92	0.92	138028	137694	137694	0	178.09
fl3795	28389	1.33	1.39	28889	28789	28772	0.059	201.39
fnl4461	181277	0.71	0.77	183688	182675	182566	0.0597	205.78
rl5915	555324	1.80	1.80	566112	565530	565530	0	278.90
rl5934	547891	1.47	1.47	558085	556085	556045	0.0071	298.09
pla7397	23123229	0.59	0.59	23421512	23261512	23260728	0.0033	310.24
rl11849	911088	1.32	1.32	929579	923288	923288	0	350.28
usa13509	19835880	0.74	0.74	19995024	19982859	19982859	0	380.24
brd14051	464229	1.10	1.10	472455	469388	469388	0	391.13
d15112	1553291	1.26	1.26	1582098	1573084	1573084	0	398.25
d18512	641889	0.52	0.56	646722	645522	645244	0.043	411.37
pla33810	65683821	0.56	0.60	66181542	66081542	66050535	0.0469	455.47
pla85900	140004561	1.67	1.77	143425209	142525209	142383704	0.0993	526.80

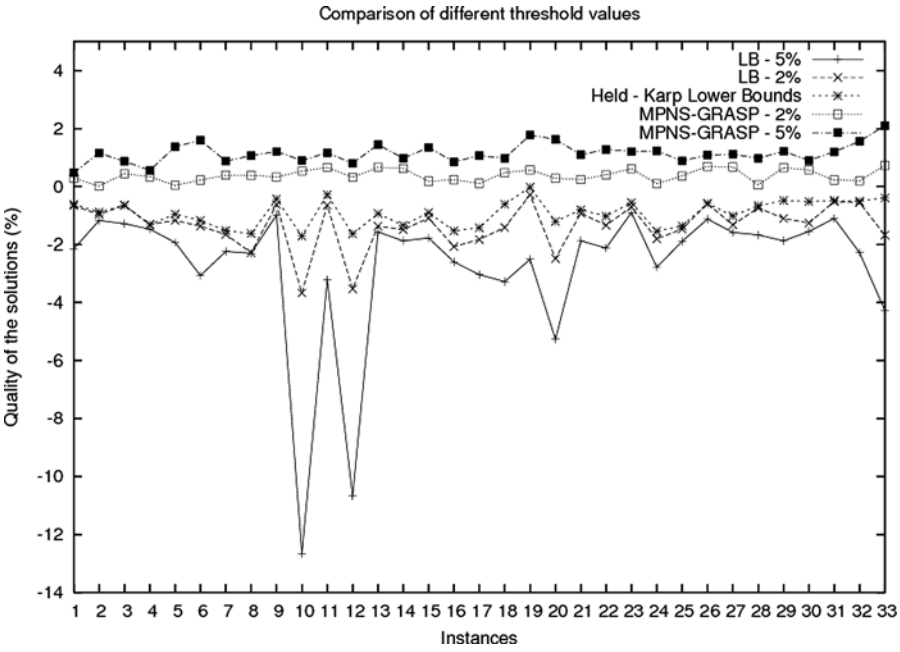

Fig. 5 Comparison of the lower bounds and the solutions for two different threshold values without using path relinking

the lower bounds and the quality of the solutions for two different threshold values without using path relinking are presented.

The x-axis gives the instances with number of nodes greater than 1000 in an abbreviated way. Thus, number 1 in the x-axis of Fig. 5 corresponds to the instance dsj1000, number 2 corresponds to the Pr1002 and so on (the instances are given in the same order as in Table 2). In the figure the quality of the Held—Karps lower bounds are also presented. As it can be seen, a threshold value equal to 2% gives very satisfactory results.

It can also be seen from Table 2 that the MPNS-GRASP algorithm without path relinking, in all instances with the number of nodes up to 280, has reached the best known solution. For the instances with the number of nodes between 299 and 2392, the quality of the solution is between 0.02% and 0.73%. For the 74 instances on which the algorithm is tested, the optimal solution is found in thirty one of them, i.e. in 41.8% of all cases. For fifteen of them a solution is found with quality between 0.02% and 0.25%, i.e. in 20.3% of all cases. For thirteen of them a solution is found with quality between 0.26% and 0.50%, i.e. in 17.6% of all cases and for fifteen of them a solution is found with quality between 0.51% and 0.73%, i.e. in 20.3% of all cases. On the other hand when the path relinking strategy is used we found the optimal solution in all but eleven instances, i.e. in 85.3% of all cases. For the other instances, the quality is less than 0.1%. It can be concluded that the Path Relinking Strategy it is very important for the MPNS-GRASP as its inclusion to the algorithm had a significant amelioration of the solutions.

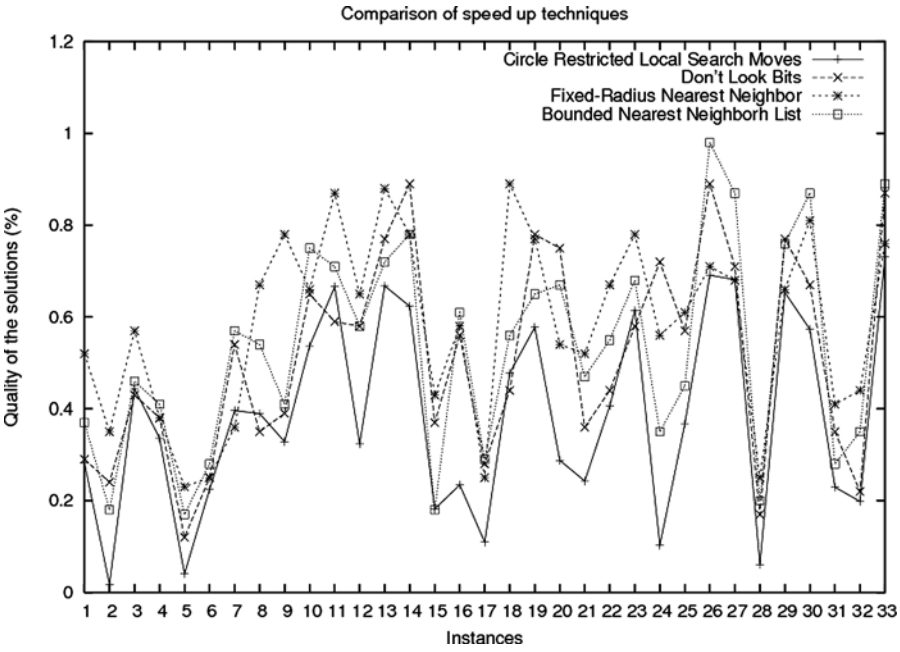

Fig. 6 Comparison between four different speed up techniques

As it has already been mentioned the most innovative technique of the algorithm is the speed up technique **Circle Restricted Local Search Moves** strategy which is a technique that is used in order to speed up the process. In order to evaluate the effectiveness of the proposed technique the local search phase of the algorithm was tested with the most common used speed up techniques like the *don't look bits technique*, the *fixed-radius nearest neighbor* and the *bounded nearest neighbor list* (Bentley 1992; Johnson and McGeoch 2002). The four different speed up techniques were tested in the instances with the number of nodes greater than 1000 because for the other instances the differences were not significant. In order to have a meaningful comparison we used the same initial solutions, the same bounds and the same local search strategies. From Fig. 6, it can be observed that the Circle Restricted Local Search Moves speed up technique is the best compared to the three other techniques, but the results are not very different between them because all the other features of the algorithms are the same.

We also test the efficiency of the Random Backtracking Lin-Kernighan (RBLK) method. This strategy was compared with the results of the classic Lin-Kernighan (LK) algorithm. In the comparison, we used as previously the same initial solutions, the same bounds and the same speed up techniques. We did not use the local search technique ENS and the path relinking strategy in our algorithm but instead we used the two different local search techniques (RBLK and LK). From Fig. 7, it is observed that the RBLK is a very efficient variant of the Lin-Kernighan algorithm which can be applied without the ENS method in the MPNS-GRASP algorithm with very good results.

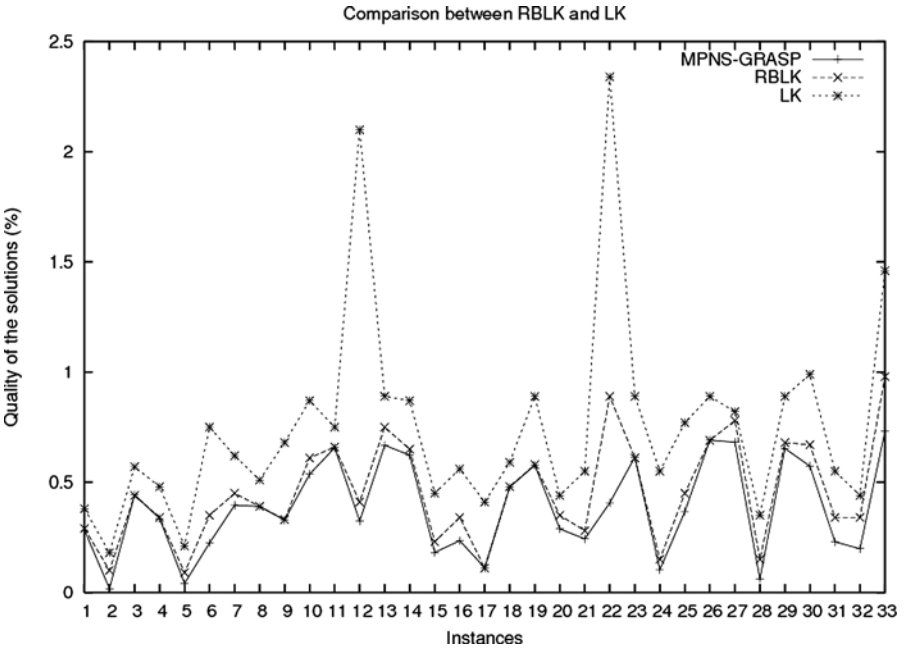

Fig. 7 Comparison between RBLK and LK

5 Conclusions

The MPNS-GRASP is a new, flexible and powerful idea for solving the Traveling Salesman Problem. The algorithm is a modification of the well known Greedy Randomized Adaptive Search Procedure. The MPNS-GRASP uses a new strategy called Expanding Neighborhood Search (ENS) for expanding the neighborhood search. This strategy is based on the Circle Restricted Local Search Moves method. The differences between GRASP and MPNS-GRASP are focused on six major issues. First, in the way of the construction of the RCL and in the way of the selection of the next candidate edge for inclusion in the tour. Second, the MPNS-GRASP has the possibility of using more than one tour construction heuristic for the initial solution of the problem. Third, the MPNS-GRASP uses the ENS strategy in the second phase. Fourth, the MPNS-GRASP has the possibility to change from one combination of tour construction heuristics and local search heuristics to others during the running of the algorithm. Fifth, the MPNS-GRASP has a termination criterion based on the existence of good lower bounds in the solution of the problem, computed with the Lagrangean Relaxation and Subgradient Optimization. Finally, the MPNS-GRASP uses a Path Relinking strategy in order to explore trajectories between elite solutions. The results of the proposed algorithm were very satisfactory.

References

Applegate D, Bixby R, Chvatal V, Cook W (2003) Chained Lin–Kernighan for large traveling salesman problems. Informs J Comput 15:82–92

Bentley JL (1992) Fast algorithms for geometric traveling salesman problems. ORSA J Comput 4:387–411

Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedure. J Glob Optim 6:109–133

Garfinkel R, Nemhauser G (1972) Integer programming. Wiley, New York

Glover F, Konchenberger G (2003) Handbook of metaheuristics. Kluwer Academic, Dordrecht

Glover F, Laguna M, Marti R (2003) Scatter search and path relinking: advances and applications. In: Glover F, Konchenberger GA (eds) Handbook of metaheuristics. Kluwer Academic, Boston, pp 1–36

Gutin G, Punnen A (2002) The traveling salesman problem and its variations. Kluwer Academic, Dordrecht

Hansen P, Mladenovic N (2001) Variable neighborhood search: principles and applications. Eur J Oper Res 130:449–467

Helsgaum K (2000) An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur J Oper Res 126:106–130

Johnson DS, McGeoch LA (1997) The traveling salesman problem: a case study. In: Aarts E, Lenstra JK (eds) Local search in combinatorial optimization. Wiley, New York, pp 215–310

Johnson DS, McGeoch LA (2002) Experimental analysis of heuristics for the STSP. In: Gutin G, Punnen A (eds) The traveling salesman problem and its variations. Kluwer Academic, Dordrecht, pp 369–444

Johnson DS, Papadimitriou CH (1985) Computational complexity. In: Lawer EL, Lenstra JK, Rinnoy Kan AHD, Shmoys DB (eds) The traveling salesman problem: a guided tour of combinatorial optimization. Wiley, New York, pp 37–85

Johnson DS, Gutin G, McGeoch LA, Yeo A, Zhang W, Zverovitch A (2002) Experimental analysis of heuristics for the ATSP. In: Gutin G, Punnen A (eds) The traveling salesman problem and its variations. Kluwer Academic, Dordrecht, pp 445–487

Laporte G (1992) The traveling salesman problem: an overview of exact and approximate algorithms. Eur J Oper Res 59:231–247

Lawer EL, Lenstra JK, Rinnoy Kan AHG, Shmoys DB (1985) The traveling salesman problem: a guided tour of combinatorial optimization. Wiley, New York

Lin S (1965) Computer solutions of the traveling salesman problem. Bell Syst Tech J 44:2245–2269

Lin S, Kernighan BW (1973) An effective heuristic algorithm for the traveling salesman problem. Oper Res 21:498–516

Marinakis Y, Marinaki M (2006) A bilevel genetic algorithm for a real life location routing problem. Int J Logist. doi:10.1080/13675560701410144

Marinakis Y, Migdalas A, Pardalos PM (2005a) Expanding neighborhood GRASP for the traveling salesman problem. Comput Optim Appl 32:231–257

Marinakis Y, Migdalas A, Pardalos PM (2005b) A hybrid genetic-GRASP algorithm using Lagrangean relaxation for the traveling salesman problem. J Comb Optim 10:311–326

Marinakis Y, Migdalas A, Pardalos PM (2007) A new bilevel formulation for the vehicle routing problem and a solution method using a genetic algorithm. J Glob Optim 38:555–580

Resende MGC, Ribeiro CC (2003) Greedy randomized adaptive search procedures. In: Glover F, Kochenberger GA (eds) Handbook of metaheuristics. Kluwer Academic, Boston, pp 219–249

Tarjan R (1983) Data structures and network algorithms. Society for Industrial and Applied Mathematics, Philadelphia