
Journal of Combinatorial Optimization, 10, 311–326, 2005

A Hybrid Genetic—GRASP Algorithm Using Lagrangean Relaxation 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 December 16, 2004; Accepted June 7, 2005

Abstract. Hybridization techniques are very effective for the solution of combinatorial optimization problems. This paper presents a genetic algorithm based on Expanding Neighborhood Search technique (Marinakis, Migdalas, and Pardalos, *Computational Optimization and Applications*, 2004) for the solution of the traveling salesman problem: The initial population of the algorithm is created not entirely at random but rather using a modified version of the Greedy Randomized Adaptive Search Procedure. Farther more a stopping criterion based on Lagrangean Relaxation is proposed. The combination of these different techniques produces high quality solutions. The proposed algorithm was tested on numerous benchmark problems from TSPLIB with very satisfactory results. Comparisons with the algorithms of the *DIMACS Implementation Challenge* are also presented.

Keywords: traveling salesman problem, genetic algorithms, metaheuristics, Lagrangean Relaxation, greedy randomized adaptive search procedure

1. Introduction

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. Heuristic attempts to solve the Traveling Salesman Problem are focused on tour construction methods and tour improvements methods. Tour construction methods build up a solution step by step, while tour improvement methods start with an initial tour and, then, try to transform it into a shortest tour. The problem with construction heuristics is that, although, they are often fast, they do not usually produce a very good solution. The most known of the tour improvement 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 produced the best results among all heuristics, is the *Lin–Kernigham heuristic* (Lin and Kernigham, 1973). Lin and Kernigham's algorithm decides dynamically at each iteration what the value of r (the number of edges to exchange) should be (Helsgaum, 2000). In the last fifteen years, a breakthrough was obtained with the introduction of metaheuristics, such as simulated annealing, tabu search, genetic algorithms and neural networks. These algorithms have the possibility to find their way out of local optima. In simulated annealing, this is achieved by allowing the length of the tour even to increase with a certain probability, while in tabu search a list of forbidden transformations is kept and so it may be necessary to use a transformation that deteriorates the objective function value in the next step. Genetic algorithms mimic the evolution process in nature. Their basic operation is the mating of two tours in order to form a new tour. In this paper a combination of genetic algorithms, GRASP and the Lagrangean Relaxation is used for the solution of the traveling salesman problem.

Genetic algorithms are randomized search techniques that simulate some of the processes observed in natural evolution (Potvin, 1996; Reeves, 1995, 2003). GRASP is an iterative two phase search which has gained considerable popularity in combinatorial optimization (Resende and Ribeiro, 2003). Each iteration consists of two phases, a construction phase and a local search procedure. Lagrangean Relaxation is a technique that produces lower and upper bounds for the solution of the problem. This lower bound is based on the construction of a minimum cost 1-tree, that is, a spanning tree with two edges incident to a node.

The proposed algorithm has a number of innovative features which will be presented in the following. The most significant of these features is that the produced lower bound is used in a stopping criterion for the genetic algorithm. This termination criterion results in a less time consuming algorithm as time is not spent in iterations with minor, if any, improvements to the solution, otherwise required in order to achieve genetic convergence.

Another innovative feature is that the initial population is produced by a Greedy Randomized Adaptive Search Procedure (GRASP) containing thus individuals of good quality.

A third feature concerns the crossover process. Although, the most popular crossover operators are the 1-point crossover, the cycle crossover and the order-based crossover, the proposed algorithm uses a new crossover operator which finds the common characteristics of the parents and inherits them to their offspring. Then a nearest neighborhood procedure is applied to each offspring in order to complete of the tour.

A fourth feature is the use of two mutation operators, that is, an attempt is made to expand the neighborhood in which the local search will be applied (Marinakis et al., 2004).

A fifth feature is that only the most promising individuals survive in the next generation, that is, an upper bound is produced by the Lagrangean Relaxation, and all the individuals whose fitness is worse than this upper bound are discarded.

The structure of the paper is as follows. In Section 2, an analytical description of the proposed hybrid genetic algorithm with Lagrangean Relaxation is presented. In Section 3, the computational results are presented and, finally, the concluding remarks are given in the last section.

2. Combined Lagrangean Relaxation and genetic algorithms for the TSP

2.1. General description of hybrid genetic algorithm

In the following, the outline of the proposed algorithm is presented.
Initialization

1. Initial computation of lower bound, upper bound and of the parameter $e = \frac{\text{upper bound} - \text{lower bound}}{\text{upper bound}} 100\%$.
2. Create the initial population of N individuals.
3. Evaluate the fitness of each individual.
4. Improve the fitness of each individual via a local search strategy.

Main Algorithm

1. Set the number of generations equal to zero.
2. Do while stopping criteria not satisfied (that is, e is less than threshold number or the maximum number of generations has been reached or genetic convergence has occured):
   - 2.1 Select the parents from the current population and choose via roulette wheel selection the pairs for mating.
   - 2.2 Apply the crossover operator between the two parents, first cloning the common features of the two parents to the offspring and then completing the offspring using a modified nearest neighborhood procedure.
   - 2.3 Improve each offspring by the two different mutation operators and insert the resulting offspring to the new population.
   - 2.4 Repeat the previous three steps until all parents are selected and mated.
   - 2.5 Rank the offsprings and the parents via their fitness function and select for the new population a number of individuals equal to the initial population.
   - 2.6 Discard from the current population the individuals that their fitness is higher than the upper bound.
   - 2.7 Calculate e. If the parameter e is less than a prespecified threshold or if the maximum number of iterations is reached or if genetic convergence has occurred, then the best individual is the final result, else update the bounds and proceed to the next generation.
3. Enddo
4. Return the best individual.

2.2. Lagrangean Relaxation for TSP

For a given set of nodes a 1-tree (Held and Karp, 1970) is a tree connecting the node set $\{2, 3, \ldots, n\}$, and having in addition two distinct arcs connecting to node 1. Therefore, a 1-tree is a graph with one cycle. The weight of the 1-tree is the sum of the cost of all its arcs. In the minimum weight 1-tree problem the objective is to find a 1-tree with minimum

weight. Such a tree can be constructed by finding a minimum spanning tree for the node set $\{2, 3, \ldots, n\}$, and by adding to it the two arcs of minimum cost incident to node 1. Any traveling salesman problem can be described as 1-tree tour in which each node has a degree 2. Thus, the minimum weight 1-tree is a lower bound (LBD) on the length of the optimal traveling salesman tour.

The traveling salesman problem is formulated as follows:

$$\text{(TSP) min cost}(T) \quad \text{where} \quad \begin{cases} T \text{ is a 1-tree with root node 1} \\ \text{degree}(v) = 2 \ \forall \text{ node } v \text{ except the root} \end{cases} \tag{1}$$

The Lagrangean Relaxation subproblem (SUB) is the following:

$$\text{SUB}(\lambda_v) = \text{min cost(T)} - \sum_{v \in N - \{1\}} (\text{degree}(v) - 2)\lambda_v, \tag{2}$$

where the λ_s for all nodes are calculated in each iteration as follows:

$$\lambda_v^{(k+1)} = \lambda_v^{(k)} + \alpha^{(k)} \frac{\text{UBD}^{(k)} - \text{LBD}^{(k)}}{||\text{degree}(v) - 2||^2} \tag{3}$$

where k is the iteration counter, UBD and LBD are the upper and lower bounds and α is a parameter in (0,2) The pseudocode which accomplishes all these calculations is given next:

Calculation of Lower Bounds
Initially values for $\lambda_v = 0$
$k = 0$
do while (Maximum number of iterations reached
or $e <$ threshold value)
Solve SUB(λ_v^k)
The cost of the SUB is a new lower bound for the TSP (NLBD)
if (degree$(v) = 2)\forall node$ **then**
The cost of the SUB is an upper bound
The algorithm stops with the optimal solution to the dual problem
else
Adjust the solution with Christofides algorithm
The cost of the Christofides is a new upper bound (NUBD)
Improve NUBD using 2-opt
endif
if NUBD $<$ UBD **then**

UBD = NUBD
endif
if NLBD > LBD **then**
LBD = NLBD
endif
$k = k + 1$
$\lambda_v^{(k+1)} = \lambda_v^{(k)} + \alpha^{(k)} \frac{\text{UBD}^{(k)} - \text{LBD}^{(k)}}{||\text{degree}(v) - 2||^2}$
$e = \frac{\text{UBD} - \text{LBD}}{\text{UBD}} 100\%$
enddo

In each iteration of the algorithm a minimum weight 1-tree is calculated and if the solution is feasible, i.e. the degree of all nodes, including the root node, is equal to 2, the algorithm stops. If a minimum weight 1-tree is a tour, w.r.t. non modified costs (i.e. $\lambda_v = 0$), then it is an optimal traveling salesman tour. When the solution is not feasible, an effort is made to convert it into a feasible one with the use of the Christofides heuristic (Lawer et al., 1985), while the cost of the 1-tree is a lower bound. In the resulting tour, a 2-opt heuristic is applied for the improvement of the solution. The cost of this tour is an upper bound. As it is desirable the lower and the upper bounds to be as good as possible, the subroutine of the main phase of the genetic is called every ten iterations. At the end of each iteration the new cost of the 1-tree is compared with the current lower bound and if the cost is higher, then the lower bound is updated. Similarly, the upper bound is updated only if the value of the cost of the current iteration is lower than the current upper bound.

2.3. Path representation

The first step in designing a genetic algorithm for a particular problem is to devise a suitable representation. Each individual is recorded based on the path representation of a tour (the sequence of the nodes). With this representation there are $2N$, where N is the number of nodes, ways to represent the same tour depending on which node is placed in position 1. In the proposed algorithm the node with number 1 is fixed to be always in the position 1 in the representation of every individual.

2.4. Initial population

Usually in a genetic algorithm there is a randomly generated initial population which may or may not necessarily contain good candidate solutions. To avoid the latter case, a modified version of the well known Greedy Randomized Adaptive Search Algorithm (GRASP) is used to initialize the population.

GRASP (Marinakis et al., 2004; Resende and Ribeiro, 2003) 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 a process which stepwise adds one element at a time to a 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 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 (c.f. multi-start local search). The pseudocode of the algorithm is presented next:

```
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
```

2.5. Calculation of fitness value

The fitness of each individual is related to the route length of each cycle. Since the TSP is a minimization problem, if a tour has a high value in the cost function then it is not a good solution for the Traveling Salesman Problem and its fitness value must be small. So a high fitness value must correspond to a tour with a small cost function. A way to accomplish this is to find initially the tour in the population with the maximum cost (length) and to subtract from this value the cost of each of the other tours. Now, the higher fitness value corresponds to the tour with the shorter length. Since the probability of selecting an individual for mating is related to its fitness, and since the individual with the worst tour has fitness equal to zero, it will never be selected for mating. To avoid this possibility the fitness of all individuals is incremented by one.

2.6. Selection probability

The parents are selected for mating via proportional selection, also known as roullete wheel selection. It is defined as follows:

1. The sum of the fitness values of all individuals is calculated.
2. A random number between zero and the sum of the fitness values is generated.
3. The intervals which will be used for the selection of the parents are determined. The first interval has a lower value equal to zero and an upper value equal to the fitness of the first individual, for the second interval the lower value is the fitness of the first individual and the upper value is the fitness of the first individual plus the fitness of the second, and so on.
4. A parent is selected based on the interval to which the random number generated in step 2 belongs.
5. To avoid the existence of a super individual (meaning an individual with a very high fitness value) which will dominate the population by its repeated selection, we impose an upper value on the times an individual can be selected.

2.7. Crossover

We propose a complex crossover operator, which initially identifies the common characteristics of the parent individuals and then inherits them to the offsprings. Subsequently, a nearest neighborhood procedure is applied to each offspring in order to complete the tour.

The common characteristics are used in the offsprings because if two or more solutions of a combinatorial problem have common characteristics there is a high probability that these also belong to the optimal solution of the problem. Also, the inheritance of good characteristics from highly adapted parents may produce even fitter offspring. For example as shown in figure 1,applying the crossover operator, the offspring has inherited the common

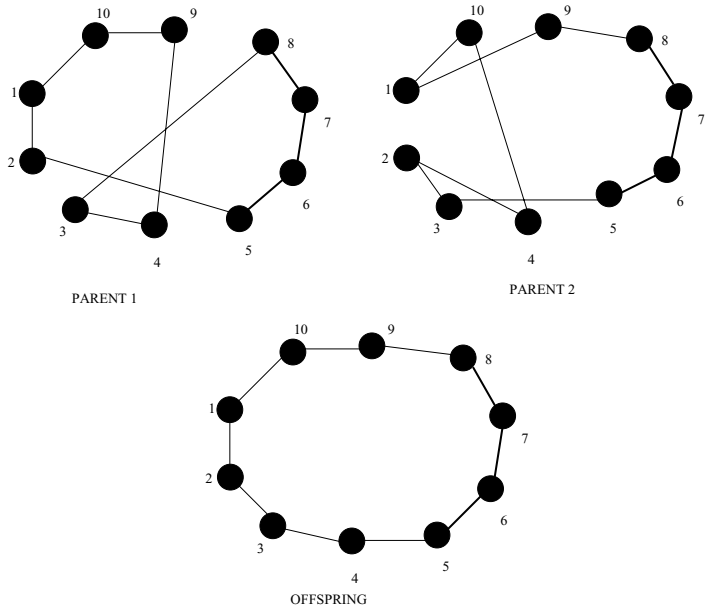


Figure 1. Example of crossover process.

characteristics of the parents (namely, the sequence of nodes 5678):

parent 1: 12**5678**34910
parent 2: 110423**5678**9
offspring: 1234**5678**910

2.8. *Mutation*

Mutation operators for the TSP are synonymous to local edge exchange heuristics. The most known of these heuristics are the 2-opt and 3-opt. Usually, only one of these heuristics is used for the mutation phase of the algorithm. In the proposed algorithm a combined neighborhood search is used. The idea of using a larger neighborhood to escape from a local minimum has been proposed initially by Garfinkel and Nemhauser (1972). First, the neighborhood function is defined as exchanging two edges of the current solution with two other edges not in the solution. This procedure is known as *2-opt procedure* and was introduced by Lin (1965) for the TSP (Lin, 1965). Subsequently, the algorithm tries to improve the solution by expanding the neighborhood using a *restricted 3-opt procedure*. The way in which 2-opt and 3-opt are used and interchanged is called Expanding Neighborhood Search by Marinakis et al. (2004). For example, in figure 2, the 2-opt mutation operator is applied to the individual 1 resulting in individual 2, while individual 3 is the result of a restricted 3-opt procedure. More precisely, in the case of 2-opt, the arcs (3, 4) and (8, 9) were replaced by the arcs (3, 8) and (4, 9). On the other hand, in the case of 3-opt, the arcs (3, 4), (6, 7) and (8, 9)

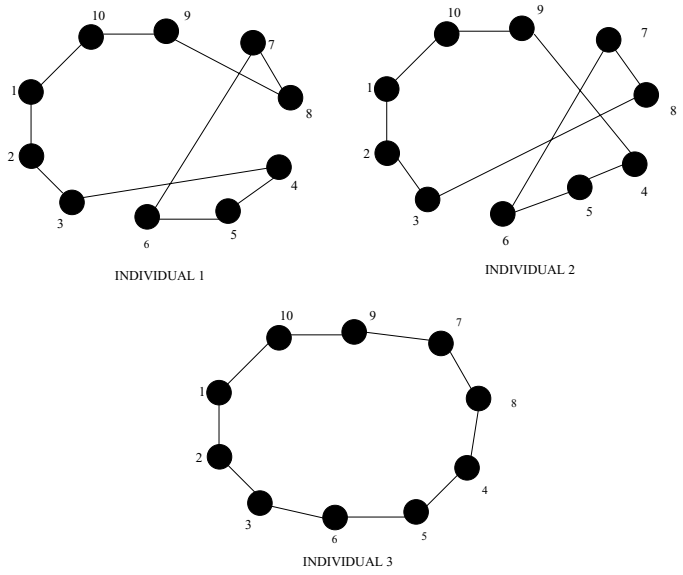


Figure 2. Example of mutation process.

were replaced by the arcs (3, 6), (4, 8) and (7, 9):

individual 1 12345678910
individual 2 123**87654**910
individual 3 123**65487**910

where the bold numbers indicate the differences in the sequence of nodes between individual 1 and individuals 2 and 3.

2.9. Population replacement

During the course of evolution, natural populations evolve according to the principles of natural selection and survival of the fittest. Individuals who are more succesfull in adapting to their environment will have a better chance of surviving and reproducing, while individuals which are less fit are eliminated. Initially, all the individuals of the population are sorted w.r.t. their fitness values. Subsequently, if their fitness is better than the Langrangean upper bound the 100 individuals with the best fitness values will replace the old population. According to the definitions given previously, upper bounds and fitness values are not comparable. Thus, a new quantity corresponding to the fitness value of the Langrangean upper bound is calculated using the procedure in Section 2.5. If the number of individuals with fitness better than the current upper bound is less than 100 then only these individuals constitute the individuals of the next generation, that is we accept a reduction in the number of individuals which constitute a population. If no individual has fitness better than the upper bound then the algorithm stops and the solution is provided by the tour implied by the current upper bound.

2.10. Termination process

In the proposed algorithm, in addition to the maximum number of generations and the genetic convergence, four different termination criteria are used. The reason for the use of the additional criteria is that the algorithm turns more efficient and less time consuming, as it is not spending time in iterations which give minor, if any, improvements to the current solution while attempting to reach pure genetic convergence. The following termination criteria are used:

a. If the upper and lower bounds from the Lagrangean Relaxation are equal, then the solution is the optimal one.
b. If the quantity $e = \frac{\text{UBD}-\text{LBD}}{\text{UBD}} 100\%$ becomes less than a threshold value, then the solution with the best fitness among all individuals is selected as a solution to the problem and the algorithm terminates. The threshold value used in the algorithm is 3%.
c. If the upper and lower bounds, the number and the fitness of the individuals remain invariable for a number of generations, the algorithm stops with the best solution found.
d. If the upper bound is better than the best fitness in the population, then the algorithm stops with a solution corresponding to the tour which provides the upper bound. If the

algorithm did not stop in such a case, then the next generation would consist of only one individual, the one implied by the upper bound.

3. Computational results

The algorithm 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 40 Euclidean sample problems with sizes ranging from 51 to 2392 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. In all tables, the first column shows the name of the instance.

In Table 1, the second and the third columns show the best lower and upper bounds produced by Lagrangean Relaxation, while the fourth column shows the parameter e. The fifth column shows the number of generations for which the algorithm runs until one of the four stopping criteria is activated. The sixth column shows the best solution found by the genetic algorithm (BSoG), the seventh column shows the best solution taken from the TSPLIB (Best Known Solution–BKS), and the last column shows the *quality* of the solution produced by the genetic algorithm. The quality of the produced solutions is given in terms of the relative deviation from the best known solution, that is $p = \frac{100(c_{\text{gen}} - c_{\text{opt}})}{c_{\text{opt}}}$, where c_{gen} denotes the cost of the best solution found by the genetic algorithm, 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 1.62%. For the 40 instances for which the algorithm was tested, the best solution was found for twenty three of them, i.e. in 57.5% of all cases, for eleven of them a solution was found with quality between 0.04 and 0.97%, i.e. 27.5%, for six of them a solution was found with quality between 1.06 and 1.62%, i.e. 15%.

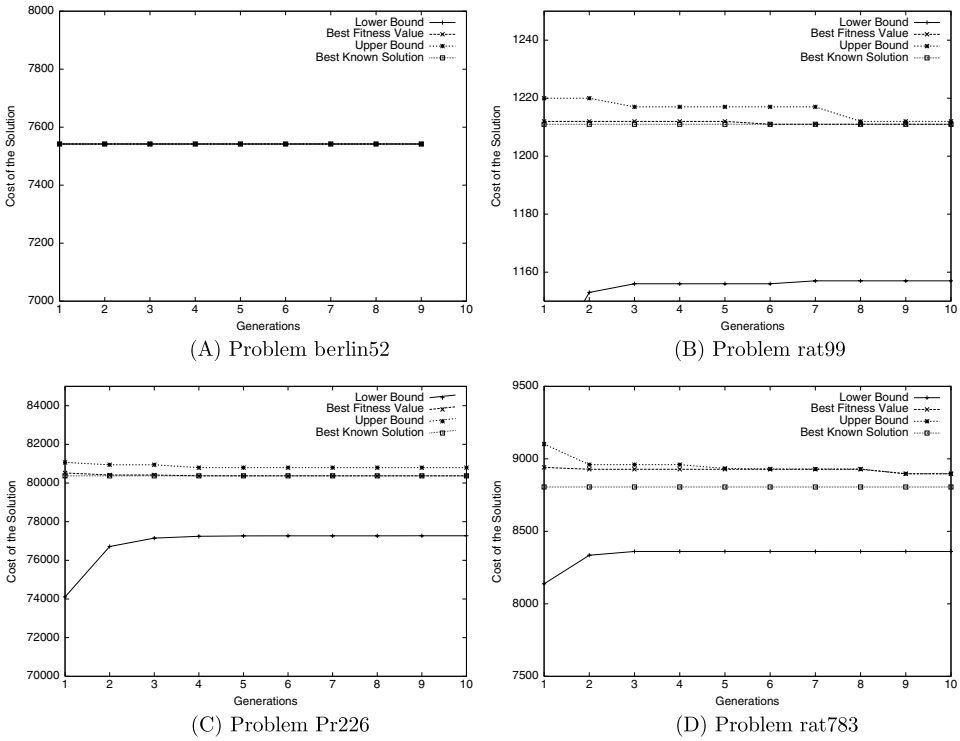
In Table 2, for four chosen instances the improvements on lower and upper bounds and best fitness value (bfv) for seven consequetive generations are presented. These instances have different features with respect to their solutions. In the first instance, namely Berlin52, after the first two generations the lower and upper bounds have the same value and, so, the optimum solution is found and the algorithm terminates. In the second and the third instances, namely Rat99 and Pr226, it is observed that the values of the lower and the upper bounds remain unchanged during a number of generations, and so, the individuals which are still active in the population do not have the possibility to improve their fitness values because the population does not accept individuals with fitness greater than the Langrangean upper bound. In the fourth instance, namely rat783, the fitness value of the best individual is equal to the Langrangean upper bound during a number of consequetive iterations, and since all the individuals with fitness greater than this upper bound are deactivated for the next generations, then, either all the active individuals have the same fitness values or there is only one individual remaining in the population. Therefore the algorithm terminates. Figure 3 depicts graphically the details in the progress of the algorithm for the examples of Table 2.

Table 1. Comparison between the solution of genetic algorithm and the best known solution.

Instance	LBD	UBD	e (%)	No. of g. − sc.	BSoG	BKS	Quality (%)
Eil51	396	432	8.33	4 − *c*	426	426	0
Berlin52	7542	7542	0	2 − *a*	7542	7542	0
Eil76	494	538	8.17	5 − *d*	538	538	0
Pr76	105090	109043	3.62	8 − *c*	108159	108159	0
Rat99	1157	1212	4.53	10 − *c*	1211	1211	0
KroA100	20901	21305	1.89	5 − *b*	21282	21282	0
KroB100	21729	22193	2.09	6 − *b*	22141	22141	0
KroC100	20432	20749	1.52	4 − *b*	20749	20749	0
KroD100	21026	21294	0.92	5 − *b*	21294	21294	0
KroE100	21740	22068	1.48	4 − *b*	22068	22068	0
Rd100	7844	7958	1.43	10 − *b*	7910	7910	0
Eil101	568	629	9.69	5 − *d*	629	629	0
Lin105	14282	14379	0.67	2 − *b*	14379	14379	0
Pr107	38930	44303	1.21	3 − *b*	44303	44303	0
Pr124	56985	59181	3.71	6 − *c*	59030	59030	0
Pr136	94578	97306	2.80	6 − *b*	96772	96772	0
Pr144	56134	58537	4.1	3 − *d*	58537	58537	0
Ch150	6404	6571	2.54	3 − *b*	6528	6528	0
KroA150	26152	26681	1.98	5 − *b*	26524	26524	0
Pr152	65244	73682	11.45	1 − *d*	73682	73682	0
D198	14038	15798	11.14	7 − *c*	15787	15780	0.04
KroA200	28956	29448	1.67	4 − *b*	29382	29368	0.04
Ts225	115504	126962	9.01	8 − *e*	126643	126643	0
Pr226	77271	80795	4.36	7 − *c*	80369	80369	0
Pr264	45324	49581	8.58	8 − *c*	49135	49135	0
Pr299	47060	48630	3.22	7 − *c*	48235	48191	0.09
Fl417	10758	11923	9.77	7 − *c*	11890	11861	0.24
Pr439	103196	107401	3.91	6 − *c*	107401	107217	0.17
Pcb442	50247	50946	1.37	7 − *b*	50946	50778	0.32
P654	31780	35132	9.54	10 − *c*	34679	34643	0.10
Pr1002	256001	262060	2.31	8 − *b*	262060	259045	1.16
Pcb1173	55793	57788	3.45	8 − *d*	57788	56892	1.57
D1291	49238	51616	4.60	9 − *d*	51616	50801	1.60
Rl1304	247273	255185	3.10	5 − *d*	255185	252948	0.88
Rl1323	263978	273115	3.34	9 − *d*	273115	270199	1.06
Fl1400	17577	20310	13.45	10 − *d*	20310	20127	0.90
Fl1577	19874	22467	11.54	8 − *d*	22467	22249	0.97
Rl1889	308286	319250	3.43	10 − *d*	319250	316536	0.85
D2103	78000	81312	4.07	7 − *d*	813112	80450	1.07
Pr2392	358092	393323	8.95	7 − *d*	384182	378032	1.62

Table 2. Improving of the upper, lower bounds and the fitness of the best individual for ten generations.

		gen.						
Instance		1st	2nd	3rd	4th	5th	6th	7th
Berlin52	LBD	7224	7542	7542	7542	7542	7542	7542
	bfv	7542	7542	7542	7542	7542	7542	7542
	UBD	7895	7542	7542	7542	7542	7542	7542
Rat99	LBD	1126	1153	1156	1156	1156	1156	1157
	bfv	1212	1212	1212	1212	1212	1211	1211
	UBD	1220	1220	1217	1217	1217	1217	1217
Pr226	LBD	74107	76716	77151	77244	77263	77267	77267
	bfv	80518	80414	80414	80369	80369	80369	80369
	UBD	81065	80942	80942	80795	80795	80795	80795
Rat783	LBD	8138	8335	8361	8361	8361	8361	8361
	bfv	8941	8928	8928	8928	8928	8928	8928
	UBD	9102	8960	8960	8960	8934	8930	8930



(A) Problem berlin52 (B) Problem rat99

(C) Problem Pr226 (D) Problem rat783

Figure 3. Improvement of the upper and lower bounds and of the fitness value of the best individual for ten generations.

Table 3. Ranking of the algorithms based on the average quality.

Rank	Method	Average
1	Tour Merging (Applegate et al., to appear)	0.004
2	ILK-NYYY (Johnson and McGeoch, 2002)	0.009
3	Iterated Lin–Kernighan by Johnson (Johnson and McGeoch, 1997)	0.041
4	Keld Helsgaun's Multi-Trial Variant on Lin–Kernighan (Helsgaum, 2000)	0.064
5	Iterated-3-Opt by Johnson (Johnson and McGeoch, 1997)	0.413
6	Augmented-LK (Johnson and McGeoch, 2002)	0.42
7	Applegate-Cook-Rohe Chained Lin–Kernighan (Applegate et al., to appear)	0.567
8	Iterated Lin–Kernighan by Neto (Neto, 1999)	0.642
9	Concorde Chain Lin–Kernighan (Applegate et al., to appear)	0.88
10	Helsgaun Lin–Kernighan (Helsgaum, 2000)	0.969
11	Tabu-search-SC-DB (Zachariasen and Dam, 1996)	1.034
12	**Hybrid genetic**	1.153
13	Expanding Neighborhood Search GRASP (Marinakis et al., 2004)	1.181
14	Lin–Kernighan-with-HK-Christo-starts (Johnson and McGeoch, 2002)	1.202
15	Tabu-search-LK-DB (Zachariasen and Dam, 1996)	1.241
16	Variable-Neighborhood-Search-Using-3-Hyperopt (Johnson and McGeoch, 2002)	1.292
17	Balas-Simonetti-Dynamic-Programming-Heuristic (Johnson and McGeoch, 2002)	1.308
18	Johnson Lin–Kernighan (Johnson and McGeoch, 1997)	1.41
19	Stem–Cycle Ejection Chain (Johnson and McGeoch, 2002)	1.702
20	LK-NYYY (Johnson and McGeoch, 2002)	1.706
21	Neto Lin–Kernighan (Neto, 1999)	1.804
22	Variable-Neighborhood-Search-Using-2-Hyperopt (Johnson and McGeoch, 2002)	1.993
23	Tabu-search-Stem - Cycle -SC (Zachariasen and Dam, 1996)	2.034
24	Tabu-search-LK-LK (Zachariasen and Dam, 1996)	2.2
25	Tabu-search-2opt-Double Bridge (Zachariasen and Dam, 1996)	3.11
26	3opt-J (Johnson and McGeoch, 1997)	3.5
27	3opt-B (Bentley, 1992)	3.635
28	Concorde-Lin–Kernighan (Applegate et al., to appear)	3.916
29	4-Hyperopt (Johnson and McGeoch, 1997)	4.062
30	Applegate Lin–Kernighan (Applegate et al., to appear)	4.308
31	3-Hyperopt (Johnson and McGeoch, 1997)	4.559
32	GENIUS (Gendreau et al., 1992)	4.685
33	Tabu-search-2opt-2opt (Zachariasen and Dam, 1996)	4.83
34	2.5opt-B (Bentley, 1992)	5.93
35	2opt-J (Johnson and McGeoch, 1997)	6.113
36	2-Hyperopt (Johnson and McGeoch, 1997)	6.218

(*continued on next page.*)

Table 3. (*continued.*)

Rank	Method	Average
37	Held Karp Christofides (Johnson and McGeoch, 2002)	6.496
38	2opt-B (Bentley, 1992)	6.83
39	GENI (Gendreau et al., 1992)	8.05
40	CCA (Johnson and McGeoch, 2002)	10.43
41	Clarke–Wright (Clarke and Wright, 1964)	11.26
42	2opt-C (Applegate et al., to appear)	14.56
43	FI (Bentley, 1992)	16.64
44	RI (Bentley, 1992)	17.47
45	Boruvka (Applegate et al., to appear)	17.76
46	CHCI (Bentley, 1992)	17.96
47	C-Greedy (Applegate et al., to appear)	18.23
48	B-Greedy (Johnson and McGeoch, 2002)	18.708
49	Q-Boruvka (Applegate et al., to appear)	19.6
50	NN (Applegate et al., to appear)	24.85
51	Spacefilling (Johnson and McGeoch, 2002)	47.545
52	Best - Way Strip (Johnson and McGeoch, 2002)	49.405
53	FRP (Bentley, 1992)	74.36
54	Strip (Johnson and McGeoch, 2002)	75.565

The results of the proposed hybrid genetic algorithm are also compared with those presented in the DIMACS Implementation Challenge (http://www.research.att.com/~dsj/chtsp/). This challenge is probably the most extensive examination to date of heuristic algorithms in the field of TSP and presents computational results for approximately 40 tour construction heuristics and for approximately 80 tour improvement heuristics. In Table 3, computational comparisons of the proposed genetic algorithm with the methods of the DIMACS Implementation Challenge are performed and the algorithms are ranked on the basis of the average quality of the solutions produced for ten instances with number of nodes between 1002 and 2392. The proposed genetic algorithm is ranked in the twelfth place among 54 algorithms. For details see (Marinakis et al., 2004) and (http://www.parallopt.tuc.gr/~yamar/,).

The proposed genetic algorithm is ranked better than all tour construction heuristics. It is also, ranked better than all simple local search algorithms with exception of two Lin–Kernighan Implementations. It is ranked better than all Variable Neighborhood Search implementations and it is also ranked better than all metaheuristics but one Tabu implementation. The proposed genetic algorithm gives slightly inferior results when compared to algorithms based on Iterated or Chained Lin-Kernighan.

4. Conclusion

The main contribution of this paper is to show that Lagrangean Relaxation can be used in hybrid synthesis with a genetic algorithm providing for the latter a qualifable termination criterion so that premature termination can be imposed on the latter in order to avoid essentially unnecessary genetic iterations which are spending computation time for minor, if any, improvements trying to achieve genetic convergence. A second contribution is the utilization of the GRASP procedure for the generation of the initial population. The technique of Expanding Neighborhood Search in the mutation phase gave very good results with respect to the quality of the solutions, resulting in a high ranking of the approach among the algorithms of the DIMACS Implementation Challenge.

References

D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "Chained Lin–Kernighan for large traveling salesman problems," *Informs Journal on Computing* (to appear).

J.L. Bentley, "Fast algorithms for geometric traveling salesman problems," *ORSA J. Computing*, vol. 4, pp. 387–411, 1992.

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.

R. Garfinkel and G. Nemhauser, *Integer Programming*, Wiley and Sons, 1972.

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.

M. Held and R.M. Karp, "The traveling salesman problem and minimum spanning trees," *Operations Research*, vol. 18, pp. 1138–1162, 1970.

K. Helsgaum, "An effective implementation of the Lin–Kernigham traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, pp. 106–130, 2000.

D.S. Johnson, "Local optimization and the traveling salesman problem," in *Proc. 17th Colloq. Automata Languages Program* Lect. Notes Comput. Sc., vol. 443, Springer-Verlag, 1990, pp. 446–461.

D.S. Johnson and L.A. McGeoch, "The traveling salesman problem: A case study," in E. Aarts and J.K. Lenstra (eds.), *Local Search in Combinatorial Optimization*, Wiley and Sons, 1997, pp. 215–310.

D.S. Johnson and L.A. McGeoch, "Experimental analysis of the STSP," in G. Gutin and A. Punnen (eds.), *The Traveling Salesman Problem and Its Variations*, Kluwer Academic Publishers Dordrecht, 2002, pp. 369–444.

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.

S. Lin, "Computer solutions of the traveling salesman problem," *Bell System Technical Journal*, vol. 44, pp. 2245–2269, 1965.

S. Lin and B.W. Kernigham, "An effective heuristic algorithm for the traveling salesman problem," *Operation Research*, vol. 21, pp. 498–516, 1973.

Y. Marinakis, A. Migdalas, and P.M. Pardalos, "Expanding neighborhood GRASP for the traveling salesman problem," *Computational Optimization and Applications* (in print), 2004.

http://www.parallopt.tuc.gr/~yamar/

D. Neto, Efficient cluster compensation for Lin–Kernighan heuristics. PhD thesis, Computer science university of Toronto, 1999.

J.Y. Potvin, "Genetic algorithms for the traveling salesman problem," *Annals of Operations Research*, vol. 63, pp. 339–370, 1996.

C.R. Reeves, "Genetic algorithms," in C.R. Reeves (ed.), *Modern Heuristic Techniques for Combinatorial Optimization*, MCGraw-Hill, 1995, pp. 151–196.

C.R. Reeves, "Genetic algorithms" in F. Glover and G.A. Kochenberger (eds.), *Handbooks of Metaheuristics*, Kluwer Academic Publishers Dordrecht, pp. 55–82, 2003.

M.G.C. Resende and C.C. Ribeiro, "Greedy randomized adaptive search procedures," in F. Glover and G.A. Kochenberger (eds.), *Handbooks of Metaheuristics*, Kluwer Academic Publishers Dordrecht, 2003, pp. 219–249.

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