Diversified Local Search for the Traveling Salesman Problem

Diptesh Ghosh
Sumanta Basu

W.P. No. 2011-01-03
January 2011

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD – 380015
INDIA

Diversified Local Search for the Traveling Salesman Problem

Diptesh Ghosh[1]
Sumanta Basu[2]

Abstract

In this paper we propose a local search based improvement heuristic called diversified local search for the traveling salesman problem. We show through computational experiments that this algorithm outperforms tabu search with similar neighborhood structures on large sized traveling salesman problem instances.
Keywords: local search; tabu search; traveling salesman problem

1 Introduction

The traveling salesman problem (TSP) is a classical combinatorial optimization problem. In this problem, we are given a complete graph $G = (V, E)$, and a cost function $c : E \rightarrow \mathbb{Z}_+$, and are required to compute a simple cycle (or tour) T in G covering all nodes in V, such that the sum of the costs of all edges in T is the minimum possible. The sum of the costs of the edges in a tour T is called the cost of the tour T. The size of a TSP is the cardinality of the node set V of the graph on which it is defined. The TSP is known to be NP-hard, and is either a model for, or is a subproblem of a large number of practical problems. The development of computational methods to solve the TSP is an active field of research, see Applegate et al. (2006) for a comprehensive review. These methods can be classified into two broad categories, exact algorithms which are guaranteed to output optimal tours, and heuristics which generate good quality tours within reasonable execution time. The former category include cutting plane algorithms (see, e.g., Dantzig et al., 1954; Grötschel and Padburg, 1985; Hong, 1972), branch and bound algorithms (see, e.g., Balas, 1965; Held and Karp, 1970; Lin, 1965), branch and cut algorithms (see, e.g., Hong, 1972; Crowder and Padberg, 1980; Grötschel and Holland, 1991; Padberg and Rinaldi, 1991) among other techniques. The latter include several construction heuristics such as the nearest neighbor heuristic, the nearest, farthest, and cheapest insertion heuristics, Christofides' heuristic (see Christofides, 1976), as well as improvement heuristics such as local search (see, e.g., Lin and Kernighan, 1973), tabu search (see, e.g., Fiechter, 1990; Gendreau et al., 1994; Knox, 1994; Potvin et al., 1996; Tsubakitani and Evans, 1998), simulated annealing (see, e,g, Cerny, 1985) genetic algorithms (see, e.g., Nguyen, 2004), and swarm algorithms (see, e.g., Goldbarg et al., 2008; Wang et al., 2003). Tabu search (see, e.g., Glover and Laguna, 1998) is the most widely applied among these heuristics. Basu and Ghosh (2008) provides a review of tabu search applications on the TSP in the literature.

Although tabu search is a most widely used heuristic to solve the TSP, it has some weaknesses, especially when implemented for TSPs of large size. First, if the search implemented with a limited execution time limit, then the quality of the tour it finally outputs depends on the initial tour given to it. If an initial tour is far from an optimal tour in the solution space, tabu search requires a large number of iterations to reach an optimal tour. Given limited execution time, this is often not possible. This weakness is partially alleviated by running tabu search in a multi-start mode,

[1]P&QM Area, IIM Ahmedabad. Email: diptesh@iimahd.ernet.in
[2]OM Group, IIM Calcutta. Email: sumanta@iimcal.ac.in

starting it from a number of initial tours distributed uniformly in the solution space. Second, once tabu search reaches a locally optimal tour, it typically requires a large number of iterations to escape the region of influence of the local optimum, i.e., to reach tours which, if provided as intial tours for local search would not converge to the local optimum that tabu search had reached. Tabu search iterations are expensive for large problems, and so a significant portion of the execution time limit provided to tabu search is used up simply to move away from any local optimum that it encounters. Third, tabu search iterations are more expensive than local search iterations on problems of the same size, since tabu has to check whether each tour obtained by performing a move on a current tour involves an edge in the tabu list.

In this paper, we describe a local search based algorithm called diversified local search (`DLS`) which addresses these weaknesses of tabu search. In Section 2 we present the `DLS` algorithm. We present our computation experience with `DLS` in Section 3 in which we compare the performance of the DLS algorithm with a tabu search algorithm for randomly generated as well as for benchmark TSP instances. We conclude the paper in Section 4 with a summary of the contributions of the paper.

2 Diversified Local Search

The diversified local search (`DLS`) algorithm is a local search based improvement heuristic. In each iteration it generates a tour at random and applies local search to obtain a locally optimal tour. It keeps track of the best locally optimal tour that it encounters, and at the end of its execution it outputs this tour as an approximation of an optimal tour for the instance.

The process of generating random initial tours is crucial, to ensure that the `DLS` algorithm does not miss out on promising regions of the solution space. We use an algorithm called `GRT` to generate such random tours. A pseudocode for the `GRT` algorithm is presented below. In the pseudocode, we assume that we have a TSP instance of size n with nodes in V labeled 1 through n, and that t initial tours have already been generated and used in the `DLS` algorithm. We represent tours as permutations of $[1, 2, \ldots, n]$.

Algorithm `GRT` (To generate a random tour)

Input: Graph $G = (V, E)$ with $|V| = n$, t tours in G.
Output: A tour T in G dissimilar to the t tours input.

Pseudocode:

Step 1: Create a permutation $\Pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of $[1, 2, \ldots, n]$. For each node in $v \in V$ set $assigned(v) \leftarrow$ FALSE. Set $k \leftarrow 1$. Go to Step 2.

Step 2: If $n = k + 1$, output T and terminate, else go to Step 3.

Step 3: Set $i \leftarrow \pi_k$. For each node $v \in V$, set $f(v) \leftarrow$ number of tours among the t tours input in which v appears in the i-th position in the tour. For each $v \in V$ for which $assigned(v)$ is TRUE, set $f(v) \leftarrow t$. Go to Step 4.

Step 4: For each $v \in V$ set $f(v) \leftarrow t - f(v)$. Let $sum \leftarrow \sum_{v \in V} f(v)$. Go to Step 5. (* Note that at the end of this step, $f(v) = 0$ for all $v \in V$ for which $assigned(v)$ is TRUE. *)

Step 5: Choose a vertex $w \in V$ at random under the condition that the probability of choosing vertex $v \in V$ is $f(v)/sum$. Assign w to the i-th position in T, set $assigned(w) \leftarrow$ TRUE, and $k \leftarrow k + 1$. Go to Step 2.

We present the pseudocode for the DLS algorithm below. Note that the GRT algorithm is used in Step 4 of the pseudocode.

Algorithm DLS (Diversified local search)

Input: Graph $G = (V, E)$, cost function $c(\cdot)$, stopping criterion.
Output: A tour *best-tour* in G.

Pseudocode:

Step 1: Set *tour* $\leftarrow$ a randomly generated tour, *best-tour* $\leftarrow$ *tour*. Go to Step 2.

Step 2: If termination condition is satisfied, then output *best-tour* and terminate. Else go to Step 3.

Step 3: Set *local-opt-tour* $\leftarrow$ tour obtained by local search starting with *tour*. If the cost of *local-opt-tour* is less than cost of *best-tour*, then set *best-tour* $\leftarrow$ *local-opt-tour*. Go to Step 4.

Step 4: Generate a random tour *tour* using the GRT algorithm. Go to Step 2.

Since GRT generates initial tours which are widely separated in the solution space, we expect the quality of the best tour encountered by DLS to improve rapidly with execution time. In Section 3 we present our computational experience with the DLS algorithm on TSP instances.

3 Computational Experience

We implemented the DLS algorithm presented in Section 2 for our computational experiments. We also implemented a 5-start tabu search algorithm, which we refer to as TS, to benchmark the performance of the DLS algorithm. Both the algorithms used the 2-opt neighborhood structure, were implemented in C, and used identical data structures wherever possible. In order to achieve a fair comparison with TS, for any TSP instance our DLS implementation was allowed to run five times with different random number seeds. Each run of the DLS algorithm and each start of the TS algorithm was given an execution time limit of 1 hour (3600 seconds).

The testbed for our comparison included both randomly generated instances and instances from the TSPLIB instance library (Reinelt, 1991). The randomly generated instances comprised of five instances each of sizes 500, 750, 1000, and 1500. These instances were complete Eulidean instances. For generating a randomly generated instance of size n, we distributed n points at random on a 1000×1000 grid, each corresponding to a node in the graph for the instance. We took the cost of the edge connecting two nodes in the graph as the Euclidean distance between the points corresponding to the node, rounded down to an integer value. This is identical to the EUC-2D distance metric in Reinelt (1991). The instances from the TSPLIB instance library included 16 instances in the library with size between 500 and 1500 which used the EUC-2D metric.

The quality of tours output by the two algorithms is measured through their suboptimality. Given a TSP instance in which the cost of an optimal tour is z^*, the suboptimality of a tour of cost z is expressed in percentage terms as $100(z - z^*)/z^*$. A lower value of suboptimality indicates a better tour. The cost of optimal tours for randomly generated TSP instances were obtained by solving the instances to optimality using the Concorde TSP Solver made available through the NEOS Server for Concorde (http://www-neos.mcs.anl.gov/neos/solvers/co:concorde/TSP.html). The costs of

optimal tours to instances belonging to the TSPLIB instance library were obtained from Reinelt (1991).

Our first observation from our computations is about the suboptimality of the best tours encountered by the two algorithms at the end of their respective execution times. The best tours obtained by the algorithms is chosen as follows. In the TS implementation, the best tour is the best among the cheapest tours encountered by the TS algorithm during its five starts. For the DLS algorithm, the best tour is the best among the cheapest tours it encounters during its five runs for a given problem.

A comparison of the suboptimality values of the best tours encountered by the two algorithms on randomly generated problems is given in Table 1. The suboptimality values for each algorithm is the average of the suboptimality values of the best tours it encounters on the five instances of a given size. The last column of the table presents a "critical time", which is the time it takes, on average, for the DLS algorithm to encounter tours which are better than the best tour encountered by the TS algorithm on a given instance. We cannot compute this value for problems of size 1500, since in two of the five instances of size 1500, the TS algorithm produced cheaper tours than thost output by the DLS algorithm.

Table 1: Suboptimality of best tours output by TS and DLS algorithms on randomly generated TSP instances

size	TS	DLS	critical time (seconds)
500	9.07%	5.36%	13.80
750	9.11%	7.14%	77.44
1000	9.44%	7.45%	385.56
1500	9.82%	8.80%	$\star$

$\star$: DLS did not encounter better tours than the best tour obtained by TS in every run.

Table 2 summarizes similar data for TSP instances from the TSPLIB instance library. The first column in the table reports the name of the instance and the second column reports its size. The third and fourth column reports the suboptimality values of the best tours obtained by the two algorithms on each of the instances. The best tours obtained by the algorithms are picked in exactly the same way as for randomly generated instances. The fifth column of the table reports the critical times for these instances.

It is clear from the table that the DLS algorithm performs better than the TS algorithm for randomly generated problem instances. The difference in the suboptimality of the best tours output by the two algorithms is smaller for larger instances. This is because the execution time limit was extremely restricting for larger problems. For randomly generated TSP instances of size 500 for example, our experiments showed that on average, the DLS algorithm could start local search from more than 2000 initial tours within one hour of execution time. For randomly generated TSP instances of size 1500, this number fell to approximately 50. We feel that given a longer execution time limit, the DLS algorithm will also outperform the TS algorithm comprehensively for larger sized TSPs.

We next report the variation in the quality of tours encountered by the DLS algorithm during its execution. For this, we examine the quality of tours obtained by the algorithm after 1 minute, 10 minutes, 30 minutes, and 1 hour of its execution. For each problem size, for each time point, we have five observations for each problem instance considered, and five problem instances. We compute the suboptimality of the 25 tours thus obtained and use their average as a measure of the quality of

Table 2: Suboptimality of tours output by TS and DLS algorithms on Euclidean instances from the TSPLIB library

instance	size	TS	DLS	critical time (seconds)
d657	657	8.92%	4.95%	16.80
d1291	1291	8.76%	7.93%	297.80
fl1400	1400	1.57%	1.22%	⋆
nrw1379	1379	9.30%	8.05%	380.20
p654	654	4.05%	1.39%	42.60
pcb1173	1173	11.33%	9.12%	196.40
pr1002	1002	8.14%	6.89%	256.60
rat575	575	8.11%	4.33%	11.40
rat783	783	8.95%	6.15%	30.40
rl1304	1304	7.93%	6.96%	343.60
rl1323	1323	9.06%	5.90%	1199.60
u574	574	7.97%	4.67%	29.40
u724	724	9.39%	6.01%	25.60
u1060	1060	8.08%	7.12%	374.60
u1432	1432	11.51%	10.00%	249.60
vm1084	1084	9.10%	6.22%	120.20

⋆: DLS did not encounter better tours than the best tour obtained by TS in every run.

tours encountered by the algorithm on problems of the given size. Table 3 presents these results for the randomly generated TSP instances that we considered in our experiments.

Table 3: Suboptimality of tours output by the DLS algorithm over time on randomly generated TSP instances

size	60s	600s	1800s	3600s
500	7.55%	6.41%	6.09%	5.90%
750	9.00%	7.93%	7.63%	7.52%
1000	9.85%	8.78%	8.32%	8.00%
1500	⋆	9.81%	9.49%	9.21%

⋆: DLS could not reach a locally optimum tour within 60 seconds

Table 2 shows that the suboptimality values for the best tour encountered by the DLS algorithm reduce as execution times are increased. Interestingly, in case of the TS algorithm, the best tour that it finally outputs is often obtained early in the execution process, and in the remaining time, the algorithm searches the neighborhood of the locally optimal tour without encountering better tours. This reinforces our conjecture that for large TSPs, the DLS algorithm is a better choice than the TS algorithm when running under execution time limits.

4 Discussion

In this paper, we introduce the diversified local search (DLS) algorithm for the traveling salesman problem (TSP). The DLS algorithm generates tours at random in the solution space of a TSP instance, and performs local search starting with the tours, keeping track of the best locally optimal

tour that it encounters. At the end of its execution, it outputs the best locally optimal tour that it encountered as an approximation to a globally optimal tour for the instance. Our computational experiments demonstrate the superiority of the DLS algorithm over a tabu search algorithm defined on the same neighborhood structure.

A close examination of the weaknesses of tabu search pointed out in the introductory section hints at why the DLS algorithm outperforms it. First, since the DLS algorithm repeatedly generates random tours from which to start local search, its output is not dependent on any initial tour input to it. Second, in Section 1 we had pointed out that in large TSP instances, if the initial tour is far from an optimal tour in the solution space, then tabu search must go through a large number of iterations before it reaches an optimal tour. In the process, it goes through improvement phases, reaches a local optimum, and then spends several iterations trying to get away from the local optimum. While the search is attempting to escape the influence of a local optimum, it spends a large number of iterations fruitlessly. In the DLS algorithm, once the search reaches a local optimum, the search moves away from it instantly by generating another random tour. If the random tour generation mechanism is one which creates random tours that are far away from other random tours generated in the history of the search, as is the case with the GRT algorithm described in Section 2, then in one iteration the search escapes the influence of the last local optimum that it encountered. Third, a DLS iteration is cheaper than a tabu search iteration when working on TSPs with the same size and neighborhood structure. This is because in a tabu search iteration, every time a move is applied on the current tour, the search must examine whether the move involves an edge which is tabu at that point. In the DLS algorithm, this check is not required, as a result, a DLS iteration requires the same amount of time required by a local search iteration.

References

D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook, (2006). The Traveling Salesman Problem: A Computational Study. Princeton Series in Applied Mathematics, Princeton University Press, Princeton, NJ.

E. Balas, (1965). An additive algorithm for solving linear programs with zero-one variables. Operations Research 13, 517–546.

S. Basu and D. Ghosh, (2008). A review of the tabu search literature on traveling salesman problems. Working Paper Series, Indian Institute of Management Ahmedabad, W.P. No. 2008-10-01.

V. Cerny, (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimisation Theory and Application 45, 41-51.

N. Christofides, (1976). Worst-case analysis of a new heuristic for the traveling salesman problem. Report No. 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.

H. Crowder and M.W. Padberg, (1980). Solving large-scale symmetric traveling salesman problems to optimality. Management Science 26, 495–509.

G. Dantzig, R. Fulkerson, and S. Johnson, (1954). Solution of a large-scale traveling-salesman problem. Operations Research 2, 393–410.

C.-N. Fiechter, (1990). A parallel tabu search algorithm for large scale traveling salesman problems. Working Paper 90/1 Department of Mathematics, Ecole Polytechnique Federale de Lausanne, Switzerland.

M. Gendreau, A. Hertz, and G. Laporte, (1994). A tabu search heuristic for the vehicle routing problem. Management Science 40, 1276–1290.

F.W. Glover and M. Laguna, (1998). Tabu Search. Kluwer Academic Publishers, Massachusetts, MA.

M. Grötschel and M.O. Holland, (1991). Solution of large-scale symmetric traveling salesman problems. Mathematical Programming 51, 141–202.

E.F.G. Goldbarg, M.C. Goldbarg and G.R. de Souza, (2008). Particle swarm optimization algorithm for the traveling salesman problem. In Greco (2008), 202–224.

F. Greco, ed., (2008). Traveling Salesman Problem. Intech.

M. Grötschel and M.W. Padberg (1985). Polyhedral theory. In Lawler et al. (1985), 252–305.

S. Hong, (1972). A Linear Programming Approach for the Traveling Salesman Problem. Ph.D. thesis. Johns Hopkins University, Baltimore, MA.

M. Held and R.M. Karp, (1970). The traveling salesman problem and minimum spanning trees. Operations Research, 18, 1138–1162.

J. Knox, (1994). Tabu search performance on the symmetric traveling salesman problem. Computers & Operations Research 21, 867–876.

E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys, eds., (1985). The Traveling Salesman Problem. John Wiley & Sons, Chichester, UK.

S. Lin, (1965). Computer solutions of the traveling salesman problem. The Bell System Technical Journal 44, 2245–2269.

S. Lin and B.W. Kernighan, (1973). An effective heuristic algorithm for the traveling-salesman problem. Operations research 21, 498–516.

D.H. Nguyen, (2004). Hybrid Genetic Algorithms for Combinatorial Optimization. Ph.D. Thesis. Department of Systems Engineering, University of Miyazaki, Japan.

M. Padberg and G. Rinaldi, (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. SIAM Review 33, 60–100.

J.-Y. Potvin, T. Kervahut, B.L. Garcia, and J.-M. Rousseau, (1996). The vehicle routing problem with time windows — Part I: Tabu Search. INFORMS Journal on Computing, 8, 158–164.

G. Reinelt, (1991). TSPLIB — A Traveling Salesman Problem Library. ORSA Journal on Computing 3, 376–384.

S. Tsubakitani and J.R. Evans, (1998). Optimizing tabu list size for the traveling salesman problem, Computers & Operations Research 25, 91–97.

K.-P. Wang, L. Huang, C.-G. Zhou, and W. Pang, (2003). Particle swarm optimization for traveling salesman problem. 2003 International Conference on Machine Learning and Cybernetics, 1583–1585.