A Fast Tabu Search Implementation for Large Asymmetric Traveling Salesman Problems

Sumanta Basu* Ravindra S. Gajulapalli† Diptesh Ghosh‡

Abstract

Real life traveling salesman problem (TSP) instances are often defined on large, sparse, and asymmetric graphs. Tabu search implementations for the TSP that have been reported in the literature almost always deal with small, dense, and symmetric instances. In this paper, we outline a tabu search implementation which can solve TSP instances much faster than conventional implementations if the graph defining the instance is sparse. We present results from computational experiments with this implementation which validate our claim.

Keywords: Tabu Search, Asymmetric Traveling Salesman Problem, Data Structures

1 Introduction

Given a graph G = (V, A, C), where V is a set of nodes, A is a set of arcs of the form (i, j), i, j V, and C = (c(i, j)) is a vector of costs where c : A $\rightarrow \mathbb{Z}_+$, the traveling salesman problem (TSP) is one of finding a simple cycle in G covering all nodes in V, such that the sum of the costs of the arcs in the cycle is the minimum possible. Cycles covering all nodes in G are called tours and the sum of the costs of the arcs in a tour is called the cost of the tour. If the existence of an arc (i, j) A implies that the arc (j, i) A and that c(i, j) = c(j, i), then the TSP defined on such a graph is called a symmetric traveling salesman problem (STSP), otherwise the TSP is called an asymmetric traveling salesman problem (ATSP).

The TSP is one of the most well-studied combinatorial problems and has a large number of practical applications (see e.g., Lawler et al. (1985)). It is also known to be NP-hard (Karp, 1972). Consequently, a lot of research e□ort has focused on optimal and heuristic algorithms to solve the TSP. In particular, in recent years, metaheuristics have been extensively used to solve this problem.

Most metaheuristics proposed in the literature are modifications of the local search algorithm. The local search algorithm is an improvement heuristic. For the TSP, it starts with a given tour, calls it the current tour, and iteratively tries to generate better tours. To do so, in each iteration, it searches a pre-defined neighborhood of tours of the current tour for the iteration, and finds the best tour in the neighborhood. If the cost of the best neighboring tour is less than that of the current tour, then the best neighboring tour is designated as the current tour for the next iteration. Otherwise, the current tour is output and the algorithm terminates. As with all local search algorithms, the local search algorithm for the TSP terminates at the first locally optimal tour it encounters. Tabu search is a metaheuristic proposed by Glover (see e.g., Glover

2 Conventional Tabu Search

Tabu search starts by designating a tour (say τ) input to it as the current tour whose neighborhood is to be searched. The best tour found by tabu search is initialized to τ , and a tabu list L is initialized to an empty list. At the beginning of each iteration, tabu search checks if pre-specified termination conditions have been met. If such conditions are met, then it terminates after printing the best tour it has found during its execution. If the conditions are not met then it executes three steps. In the first step, it searches the neighborhood of the current tour τ ac

3.1 Data structures to store graphs

Complete graphs are predominantly used in tabu search literature and therefore the most e□cient data structure for storing the costs of edges or arcs is the adjacency matrix. Given a TSP with n nodes, this is a $n \times n$ matrix $A = [a_{ij}]$, in which a_{ij} stores the cost of the arc from node i to node j. In case the TSP is symmetric, it is su□cient to store a_{ij} values only when $i < j$. If an arc (p, q) does not exist in the graph then a_{pq} is set to .

If the graph is sparse, storing costs of infeasible arcs is ine□cient. Papers like Saad (1994) and Eijkhout (1992) elaborate on di□erent data structures used to store sparse matrices, of which the compressed row format is especially e□ective. In our tabu search implementation, we use a minor modification of this format to store the arc costs for a given problem.

Given an asymmetric graph $G = (V, A, C)$, we define an array `arcs` and a vector `arc_position_ptr`. Without loss of generality, let us assume that the nodes in the graph are numbered 1 through n and $|A| = m$. `arcs` is a $m \times 3$ matrix, in which each row represents one arc in A. The first column stores the tail of an arc, the second column stores its head, and the third column its cost. The arcs are ordered in non-decreasing order of their tail nodes, and then according to the increasing order of their head nodes. `arc_position_ptr` is a vector of size n. The i-th element of `arc_position_ptr` stores the smallest row number in `arcs` which represents an arc with i as its tail. Figure 1 presents an example of the data structures that we use to store graphs in our implementation.

are reversed. Hence, in our implementation, we store the tour as a $n \times 6$ array `tour_arcs` where each row corresponds to data about one arc in the tour. Consider a row in the array which corresponds to arc (i, j). Columns 1 and 2 store i and j, and column 3 stores $c(i, j)$. Column 4 stores a value of 1 if arc (j, i) exists in the graph, and 0 otherwise. Column 5 stores the value of $c(j, i)$ if it exists, and column 6 stores a value of 1 or 0 depending on whether the arc (j, i), if it exists, is in the tabu list or otherwise. Notice that the information being stored in the fourth and fifth columns can be obtained by looking up the graph data structure, but by storing them in the tour data structure, we can obtain these in constant time rather than spend O(log(k)) time to look it up from the graph data structure. We also store a $n \times 2$ array called `tour_node_ptr`. The first column of the i-th row of this array stores that row in the `tour_arcs` matrix which has node i as its tail. The second column stores that row in the `tour_arcs` matrix which has node i as its head.

Example: **Consider the tour $1 \rightharpoonup 2 \rightharpoonup 3 \rightharpoonup 4 \rightharpoonup 5 \rightharpoonup 6 \rightharpoonup 1$ in the graph in Figure 1. Assume that**

tail	head	arc cost	reversible?	rev. arc cost	rev. arc tabu?
1	2	3	1	2	0
2	3	1	1	6	1
3	4	4	1	5	1
4	5	1	0	*	*
5	6	5	1	3	0
6	1	9	0	*	*

`tour_arcs`

tail	head
1	6
2	1
3	2
4	3
5	4
6	5

`tour_node_ptr`

Figure 2: Representation of a tour in our implementation

the tabu list is $\{(2,6), (3,2), (3,5), (4,3)\}$. Figure 2 represents the tour structure used to store the tour in our implementation. In the representation a '*' at any position implies that the value at that position is not important for storing tour information. For example, for arc (4,5) represented

large sparse ATSPs, the computation of flag values require O(n) time.

At the end of the flag value computation, let us assume that we have a list of K ($\ll n-3$) arcs which yield feasible tours through a 2-opt operation with a_i. Finding the best among these K tours takes O(nK) time, which can be reduced through intelligent book-keeping. Since there are n arcs in a tour Algorithm 1 requires $O(n^2 K_{avg})$ time, where K_{avg} denotes the average number of arcs that can participate in a 2-opt operation with any given arc in a tour. Note that in a conventional implementation, the equivalent time complexity is $O(n^3)$. The effect of this decrease in time required to perform tabu search iterations is evident from the results presented in Section 4.

4 Comput

asymmetric graphs TS-SAG**, and an implementation of conventional tabu search using the 2-opt neighborhood** TS-CI**. We coded a third implementation of conventional tabu search using the irreversible 3-opt neighborhood structure and called it** TS-3OPT**. All the three implementations only used tabu lists, which had a capacity to store 50 arcs, and no intermediate or long term memory structures. Our tabu list is longer than the usual size of such lists reported in the literature. This is because, from preliminary experiments we have observed that shorter tabu lists lead to cycling in tabu search implementation for the problem sizes that we consider in our experiments. The algorithms were coded in C and the experiments were performed on a computer with 3 GB RAM and a 2.4GHz Quad Core processor.**

The experiments were divided into two parts. In the first part, we allowed the implementations to run for a pre-specified number of tabu search iterations, and in the second part all the implementations were allowed to run for a pre-specified amount of execution time. We describe the experiments and results in detail in the remainder of this section.

Experiments in which the number of tabu search iterations were fixed

For these experiments we generated instances on graphs with 1000, 2000, and 3000 nodes with densities of 0.01, 0.02, and 0.05. We proposed to allow each implementation to run for 1000 tabu search iterations. However from preliminary experiments, we observed that TS-3OPT **took extremely**

Experiments in which the execution times were fixed

For these experiments we chose the same instances that we used for the first part of the experiments. In this part, we allowed each run of each of the three implementations to execute for one hour, i.e., 3600 seconds. Obviously, this means that the number of tabu search iterations allowed by each of the implementations are di□erent. The measure of performance for this part of experiments was the relative quality of the tours that the implementations output after one hour of execution.

Since the cost of tours output by the same implementation for di□erent runs are di□erent for the same instance, our method of evaluating the performance of the three implementations is the following. Let us suppose that for a given ATSP instance and a given starting tour, TS-CI**,** TS-SAG**, and** TS-3OPT **output tours with cost c_c, c_s, and c_3 respectively. We define the relative quality of the tour output by** TS-CI **as $q_c = 1$, that of the tours output by** TS-SAG **and** TS-3OPT **as $q_s = c_s/c_c$ and $q_3 = c_3/c_c$ respectively. For the given instance, the relative quality**

References

Basu S and Ghosh D. (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.**

Brando J and Mercer A (1997). A tabu search algorithm for the multi-trip vehicle routing and scheduling problem. *European Journal of Operational Research* 100:**180–191.**

Cordeau J, Gendreau M and Laporte G (1998). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks* 30:**105–119.**

Cordeau J, Laporte G and Mercier A (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *The Journal of the Operational Research Society* 52:**928–936.**

Eijkhout V (1992). *Distributed sparse data structures for linear algebra operations.* **Technical report, Computer Science Department, University of Tennessee, Knoxville, TN.**

Glover F (1989). Tabu search– part I. *ORSA Journal on Computing* 1:**190–206.**

Glover F (1990). Tabu search– part II. *ORSA Journal on Computing* 2:**4–32.**

Glover F (1996). Finding a best traveling salesman 4-opt move in the same time as a best 2-opt move. *Journal of Heuristics* 2:**169–179.**

Glover F and Laguna M (1998). *Tabu Search*. **Kluwer Academic Publisher.**

Golden B, Wasil E, Kelly J and Chao I (1998). Fleet management and logistics. In: *The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results*. **pp 33–56. Kluwer Academic Publishers.**

Homberger J and Gehring H (2005). A two-phase hybrid metaheu