INDIAN INSTITUTE OF MANAGEMENT CALCUTTA

WORKING PAPER SERIES

WPS No. 735/ October 2013

Preprocessing Schemes for Tabu search on Asymmetric Traveling Salesman Problem

by

Sumanta Basu
Assistant Professor, IIM Calcutta, D. H. Road, Joka P.O., Kolkata 700 104 India

&

Megha Sharma
Assistant Professor, IIM Calcutta, D. H. Road, Joka P.O., Kolkata 700 104 India

Preprocessing Schemes for Tabu search on Asymmetric Traveling Salesman Problem

Sumanta Basu[*] Megha Sharma [†]

Abstract

The objective of this paper is to implement tabu search on moderate sized asymmetric traveling salesman problems (ATSPs). We introduce preprocessing schemes based on the cross entropy and the particle swarm optimization method which allow us to reduce the number of arcs in the graph defining an ATSP instance, without significantly affecting the cost of the tour output by tabu search. This reduction helps us to apply tabu search methods especially designed for ATSPs defined on sparse graphs. We also provide a scheme to generate good initial tours for multi-start tabu search to run on large problems. We report our computational experiences on randomly generated problems as well as benchmark problems to show that our method yields good quality tours for moderate sized ATSPs much faster than conventional tabu search implementations.

Keywords: traveling salesman problem, tabu search, cross entropy, particle swarm optimization

1 Introduction

The Traveling Salesman Problem (TSP) is an extensively studied class of combinatorial optimization problems which finds its application primarily in modeling the problems arising in transportation sector. Minor variations of TSP are also used for modeling problems in other domains such as scheduling, project planning etc. (Tang et al., 2000). Tabu search (Glover, 1989, 1990) is a local search based metaheuristic which has been extensively applied to solve a wide variety of combinatorial optimization problems. Since its inception, many successful applications of tabu search on TSP have been reported in the

[*]OM Area, Indian Institute of Management Calcutta. Email: sumanta@iimcal.ac.in
[†]OM Area, Indian Institute of Management Calcutta. Email: megha@iimcal.ac.in

published literature (Knox, 1994; Fiechter, 1994). Although tabu search has been able to provide reasonably good results on TSPs for the last two decades, its success is limited to smaller problem instances. Therefore, in this paper we present a novel preprocessing method for applying tabu search on large instances of TSP. Our preprocessing method reduces the computational time for large instances with a minor sacrifice in the solution quality. Our work is particularly suitable for those applications where one needs to repeatedly solve many TSPs and for each TSP one requires a reasonably good quality solution in relatively less time.

Consider a digraph $G = (V, A)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n nodes, and $A = \{(v_i, v_j)|v_i, v_j \in V\}$ is a set of arcs. Each arc (v_i, v_j) has a cost c_{ij}. Density of graph G is then defined as $\rho = |A|/\{|V| \times (|V| - 1)\}$. If $\rho = 1$, the graph is called complete, and if ρ is significantly less than 1, the graph is called sparse. If the existence of an arc (v_i, v_j) in A implies that (a) $(v_j, v_i) \in A$, and (b) $c_{ij} = c_{ji}$, then the graph is called symmetric, otherwise it is called asymmetric.

A tour in G is a simple directed cycle covering all nodes in V. The cost of a tour is the sum of costs of all the arcs in the tour. The traveling salesman problem is one of finding a minimum cost tour in G. The cardinality n of V is called the size of the TSP. If a TSP is defined on a symmetric digraph, it is called a symmetric traveling salesman problem (STSP), otherwise it is called an asymmetric traveling salesman problem (ATSP). This paper focuses on developing fast implementations of tabu search for large instances of ATSPs.

Since the Hamiltonian Cycle problem is $\mathscr{NP}$ complete (see Karp, 1972) and hence the TSP is $\mathscr{NP}$ hard, both exact algorithms (for an overview, see e.g., Applegate et al., 2006; Fischetti et al., 2002; Roberti and Toth, 2012; Laporte, 2010) and heuristic algorithms (for an overview, see e.g., Johnson et al., 2002; Laporte, 2010) have been used in the literature to solve these problems. Success of exact algorithms is limited to smaller problem sizes as the computational time becomes prohibitive on larger instances. Prohibitive computational time of larger instances explains the proliferation of heuristics in published literature in recent years. The issue of excessive computational time and therefore limited applicability of exact algorithms becomes more relevant in the context of ATSP due to its more complex nature.

In practice, for example in logistics, ATSPs occur more commonly than STSPs. However, most of the research in TSP has focused on the STSP. For STSPs, researchers have been successful in obtaining results which allow us to solve most of the large instances in very little time. However, such results are not available for ATSPs. According to Johnson et al. (2002), a possible reason for this could be the absence of a general underlying structure for ATSP instances which could enable algorithms to reduce computational time by exploiting specific problem characteristics. One line of approach for solving ATSP attempts to convert an ATSP instance into an equivalent STSP instance to obtain solutions (see, e.g., Cirassela et al., 2001). This approach has limited success since the size of the equivalent STSP instance is typically much larger than the size of the original ATSP instance. Even implementation of metaheuristics on ATSP instances also has limited evidence of success in published literature. In this paper, we present tabu search based solution method for moderate sized ATSP instances. We chose tabu search as the base of our solution method as from the published literature it appears to be one of the most successful metaheuristics for TSP. While tabu search has been widely applied to TSPs, most of the literature on tabu search for the TSP is restricted to the STSP (Basu, 2012). Even ATSP instances on which tabu search implementations have been reported in the literature are quite small, mostly restricted within 500 nodes.

In this paper we develop preprocessing schemes for tabu search implementations that enable us to apply tabu search on ATSP instances larger than those reported previously in the literature in significantly shorter computational time. Evidence of combining preprocessing schemes with exact algorithms is present in published literature but similar idea is rarely implemented in the context of metaheuristics. Our idea is novel in that sense and is relevant because of failure of traditional metaheuristic implementations on large problem sizes. To solve large instances of ATSPs, we use the tabu search implementation presented in Basu et al. (2013) which speeds up tabu search significantly on ATSP instances defined on sparse graphs. Note that this method has been designed for ATSP instances defined on sparse graphs. In this work, we focus on ATSP defined on complete graphs. Therefore, to exploit the niceties of the tabu search implementation presented in Basu et al. (2013), we devise schemes that reduce the density of the graph underlying an ATSP instance. These reduction schemes, presented in Section 2, may eliminate an optimal tour, but this fact is not of much concern to us as we plan to use a tabu search algorithm which is itself not guaranteed to output an optimal solution.

It is interesting to note that there is almost no literature on preprocessing of TSP instances. On the contrary, preprocessing algorithms exist for other well-studied combinatorial problems, see for example, Goossens and Baruah (2001) for uniprocessor scheduling, and Khumawala (1975) for the uncapacitated facility location problem. While Toth and Vigo (1998) and Glover and Laguna (1998) describe approaches that can be used for preprocessing, none of these papers explicitly describes a preprocessing procedure. For preprocessing, we chose two commonly used heuristics in recent times: cross entropy and particle swarm optimization. Both cross entropy and particle swarm optimization are population based heuristics which attempt to make balance between search intensification and search diversification.

Tabu search, when designed to solve large problems is often implemented in a multi-start manner, where it is run from multiple starting points which are suitably separated in the solution space. The best solution encountered by tabu search among all the runs is output as the final solution. In designing a multi-start tabu search implementation, one needs to ensure that the starting solutions are widely dispersed in the solution space. In Section 3, we describe ways in which the preprocessing schemes described in Section 2 can be modified to generate suitable initial tours for multi-start implementations of tabu search.

In Section 4 we describe the results of our computational experience with the proposed preprocessing schemes on randomly generated ATSP instances as well as on benchmark problem instances. The randomly generated instances used in our experiments include problem instances of sizes 200 to 600. The benchmark instances were obtained from the collection of ATSP instances from Johnson et al. (2002). With our proposed preprocessing schemes and initial tour generation methods, we created five different tabu search implementations in such a way that comparisons among the implementations provide us with information about the efficiency of our preprocessing and initial tour generation methods. We obtained quite encouraging results from our experiments. Our computational experiments recommend the use of cross entropy based preprocessing scheme as it outperforms the conventional tabu search implementation as well as the particle swarm optimization based preprocessing scheme convincingly when solution quality and computational times are compared. For some benchmark problem instances, it outperforms some of the best heuristics described by Johnson et al. (2002) by reducing computational time substantially with a minor decrease in solution quality. We summarize the findings of our computational experiments in Section 5. We also present a summary of the contribution of this paper along with directions for future research in this section.

2 Preprocessing Schemes

Consider an ATSP instance defined on a complete graph. Within its set of arcs, some are either too costly or too inconveniently located to be included in any low cost tour. Preprocessing is a process which attempts to eliminate such arcs from the graph. The sparse graph thus obtained can then be addressed efficiently using tabu search implementations designed especially to work on sparse graphs (i.e., Basu et al., 2013). The implementation proposed in Basu et al. (2013) exploits the fact that the neighborhood of a tour in a sparse graph is much smaller than the neighborhood of the same tour in a complete graph. In the remainder of this section, we describe two different preprocessing algorithms which we have developed for sparsifying the complete graphs underlying the ATSP instances. The preprocessing algorithms are based on cross entropy (CE) and particle swarm optimization (PSO), two heuristics with diverse mechanisms to explore the solution space.

2.1 Cross Entropy

Cross entropy was developed as a tool for rare event simulation in Rubinstein (1997). This tool was later used in Rubinstein (1999, 2001) to solve combinatorial optimization problems. CE has been used for the TSP in Chepuri and Homem-de Mello (2005); Boer et al. (2005). It is an iterative process in which arcs that are less likely to be in good quality tours are progressively eliminated until only those arcs that are in the "best" tour remain. We use a truncated CE algorithm to reduce the number of arcs in a graph defining a TSP instance.

In our preprocessing algorithm, we take a directed graph $G = (V, A)$ along with three parameters k, e, and *iter* as input. We define a probability matrix $P = [p_{ij}]$, where p_{ij} denotes the probability that the arc (v_i, v_j) will be included in a random tour during a particular iteration.

At the beginning of the preprocessing algorithm the matrix $P = [p_{ij}]$ is initialized as $p_{ij} = 1/(n-1)$ and $p_{ii} = 0$ for all $i, j \in \{1, \ldots, n\}$, $i \neq j$ and $|V| = n$. The matrix P is updated after each iteration of the algorithm. A typical iteration starts with the probability matrix P. During the iteration, we generate k tours in G such that the probability of an arc (v_i, v_j) being chosen in a tour is proportional to p_{ij}. We then create an elite tour list $\mathscr{E}$ containing the e lowest cost tours from among the k tours generated above. At the end of the iteration, we update P as follows. Let n_{ij} be the number of times that arc (v_i, v_j) appears in the tours in $\mathscr{E}$. Then $p_{ij} = n_{ij}/e$. After *iter* iterations get over, a sparse graph $G' = (V, A')$ is formed where $A' = \{(v_i, v_j) : p_{ij} > 0\}$.

From preliminary experiments we observed that in some cases at the end of the specified number of iterations, the output graph became too sparse and as a result some of the tours did not have any neighboring tour. In such graphs, tabu search was unable to better the best tour in the $\mathscr{E}$ list output after *iter* iterations. So motivated by Toth and Vigo (1998), we added a step in our algorithm in which we chose a threshold τ, and added those arcs in A whose costs were less than the threshold τ to A'. The value of τ was chosen as 1.5 times the average cost of the arcs present in the tours in set $\mathscr{E}$ at the end of the first iteration based on preliminary experiments. The result of this operation is a denser G' but one in which some neighboring tours do exist for most of the tours. Algorithm 1 describes the set of steps required for preprocessing by cross entropy.

To generate a random tour using P matrix, it takes $\mathscr{O}(n^2)$ time. Moreover we require to create k such tours to populate initial tour set K and hence the computational complexity of Step 8 in Algorithm 1 becomes $\mathscr{O}(k.n^2)$. Finding out best e tours from set K requires $\mathscr{O}(klogk)$ time using heap sort technique. Also the update of probability matrix P in Step 11 takes $\mathscr{O}(k.n^2)$ time. Hence the computational complexity of Algorithm 1 is $\mathscr{O}(k.n^2)$.

Algorithm 1 Preprocessing by Cross Entropy

```
Input:  A complete graph $G = (V, A)$, $k$, $e$, $iter$

Output: Graphs $G' = (V, A')$ with $|V| = n$ and $|A'| \leqslant n(n-1)$

 1: INITIALIZATION
 2: Initialize the parameter values and the probability matrix $P$
 3: TERMINATION
 4: If the number of iterations reaches $iter$, Go to Step 5, Else Go to Step 7
 5: Terminate the program and
 6: If (Arc $a \in A$ appears within tours of set $E$ or cost($a$) $< \tau$), Include arc
    $a$ in $A'$
 7: ITERATION
 8: Generate set $K$ of $k$ feasible tours using probability matrix $P$
 9: Find the best $e$ tours from set $K$ to form set $E$
10: IF ($iter = 1$), then find threshold $\tau$ from tours in set $E$
11: Update probability matrix $P$ using tours in set $E$
12: Go to Step 3
```

2.2 Particle Swarm Optimization

Particle swarm optimization was first proposed by Kennedy and Eberhart (1995) for continuous optimization problems. Kennedy and Eberhart (1997) later developed a variant for discrete optimization problems. Given a graph $G = (V, A)$, a typical PSO starts by generating a number (called the swarm size and denoted by PSO_{SS}) of feasible tours. The set of these feasible tours is called the swarm and each tour in the swarm is called a particle. Each tour i in the swarm is uniquely represented by its position x_i and has two associated characteristic called its velocity v_i, its personal best position

p_i which stores the lowest cost tour that this particle has achieved so far. For the whole swarm, we also store the global best position p_g which stores the best tour found by the PSO so far. At the beginning of the algorithm, the personal best position p_i for each particle i is initialized by its current position and the global best position p_g for the swarm is initialized by the best position in the swarm. At each iteration of the algorithm, the velocity v_i for each particle is updated based on its current position, its personal best position and the global best position. The position of a particle in the next iteration is obtained based on its velocity. Based on the new position of the particles in the swarm, their personal best positions, and the global best positions are updated, if the new positions are better than the previous ones. The process is repeated for a prespecified number of iterations PSO_{iter}. The global best position at the end of PSO_{iter} iterations is output as the solution by the algorithm.

For our preprocessing scheme, we follow the discrete PSO algorithm for TSP developed by Bin et al. (2012). The algorithm developed in this paper, uses binary strings to denote the cities in a tour. Each tour is represented as $x = x_{11}, \ldots, x_{1c}, x_{21}, \ldots, x_{2c}, \ldots, x_{n1}, \ldots, x_{nc}$, where n is the number of cities and c is the smallest constant with $2^c \geqslant n$. All the operations such as velocity determination and position updation for the particles are applied on these binary strings. These strings are converted back to the sequence of visited cities using Algorithm 2.

Algorithm 2 Conversion of cities in TSP from binary representation to a sequence of cities

```
1: Tour $T = \{1, 2, ..., n\}$
2: Corresponding binary representation $X = x_{11}, ..., x_{1c}, x_{21}, ..., x_{2c}, ..., x_{n1}, ..., x_{nc}$
3: For i=1 to n
4: Convert the binaries to decimal
5: While $(X > |T|)$
6: $X = X - |T|$
7: end While
8: $S_i = C_X$; $S_i$ is the $i$th visited sequence and $C_X$ is the converted decimal number from binary representation
9: $C = C - C_X$; remove city $S_i$ from the candidate set
10: end
```

Bin et al. (2012) introduced the concept of leader tours (x_i^L) by identifying some elite tours in each iteration. The number of elite tours is restricted by a parameter PSO_{lead} and the rest of the tours in swarm are recognized as followers (x_i^F). Leaders' positions are updated using Equations 1 and 2.

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_1 (p_{ij}(t) - x_{ij}^L(t)) + c_2 r_2 (p_{gj}(t) - x_{ij}^L(t)) \tag{1}$$

$$x_{ij}^L = \begin{cases} 1, & \text{if } rand() < S(v_{ij}) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $S(v_{ij}) = \frac{1}{1+e^{-v_{ij}}}$ and $rand()$, r_1, r_2 are a random numbers selected from a uniform distribution in $[0, 1]$. c_1 and c_2 define the degree of influences of p_i and p_g respectively on particle's velocity. jth bit of ith tour is represented as x_{ij} with velocity as v_{ij} bounded within a range of $[V_{min}, V_{max}]$ to prevent large fluctuation of the solution during search process. Follower's positions are updated using Equations 3 and 4.

$$prob_{ij}^F(t+1) = \begin{cases} \min(prob_{ij}^F(t) + \frac{1}{\alpha_{LIF}}, 1), & \text{if } \frac{\sum_{i=1}^{PSO_{lead}} x_{ij}^L(t)}{PSO_{lead}} \geqslant 0.5 \\ \max(0, prob_{ij}^F(t) - \frac{1}{\alpha_{LIF}}), & \text{otherwise} \end{cases} \tag{3}$$

$$x_{ij}^F(t+1) = \begin{cases} 1, & \text{if } rand() < prob_{ij}^F(t+1) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

As can be seen from Equation 3, the probability $prob_{ij}^F(t+1)$ that the bit x_{ij}^F will be set to 1 in iteration $t+1$ depends on the values of that particular bit for all the leaders, leader impact factor α_{LIF}, and the value of this probability in the previous iteration. Equation 3 implies that if the value of a particular bit j is equal to 1 for at least half of the leaders in the tth iteration then the follower's corresponding bit will have a higher probability to be set to 1 in $(t+1)$th iteration. If the global best solution does not change from one iteration top another, mutation is applied to all the tours in the swarm using mutation probability PSO_{mp} as described in Equation 5.

$$x_{ij}(t+1) = \begin{cases} 1 - x_{ij}(t+1), & \text{if } rand() < PSO_{pm} \\ x_{ij}(t+1), & \text{else} \end{cases} \tag{5}$$

We use this PSO algorithm to sparsify the complete graph underlying an ATSP instance. In our preprocessing scheme, the PSO is run for a pre-specified number of iterations, PSO_{iter}, to generate a reduced graph $G = (V, A')$. The arc set A' of the reduced graph is constructed by adding all those arcs whose frequency in the leader tours across all iterations is more than a threshold value ($PSO_{thresh-lead}$). The density of the reduced graph

Algorithm 3 Preprocessing by PSO

Input: A complete graph $G = (V, A)$, PSO_{SS}, PSO_{iter}, $PSO_{\alpha_{LIF}}$, PSO_{mp}, PSO_{lead}, $PSO_{thresh-lead}$, PSO_{best}

Output: Graphs $G' = (V, A')$ with $|V| = n$ and $|A'| \leqslant n(n-1)$

```
 1: INITIALIZATION
 2: Initialize the parameter values
 3: ITERATION-STEP 0
 4: Generate $PSO_{SS}$ initial feasible tours and calculate the fitness value
    (length) of each tour
 5: Identify $PSO_{lead}$ leaders based on fitness values
 6: Initialize personal best tour for each particle and the global best tour
 7: Store $PSO_{best}$ tours by sorting all the particles based on their tour lengths
 8: TERMINATION
 9: Terminate the program if the number of iterations reaches $PSO_{tier}$, Else
    Go to Step 10
10: ITERATION
11: Update leader positions using Equations 1 and 2
12: Update follower positions using Equations 3 and 4
13: If no change in global best is found, update leader and follower using
    Equation 5
14: Update global best and the best tour in each position
15: Update $PSO_{best}$ tours, if required
16: Go to Step 8
```

is further increased by adding all those arcs that have ever appeared in the best PSO_{best} number of tours in any iteration.

Before initializing PSO preprocessing algorithm, conversion from decimal to binary sequence of cities to form a tour takes $\mathscr{O}(nc)$ time. Updating leader and follower tours in each iteration requires $\mathscr{O}(nc.PSO_{lead})$ and $\mathscr{O}(nc.(PSO_{SS} - PSO_{lead}))$ respectively. Hence the overall computational complexity of Algorithm 3 is $\mathscr{O}(nc.(PSO_{SS} - PSO_{lead}))$ as $(PSO_{SS} - PSO_{lead}) > PSO_{lead}$.

3 Generation of Initial Tours

Performance of any local search heuristic like tabu search is often critically dependent on the quality of the solution used as a starting point for the algorithm. As the problem size increases, the solution space increases exponentially, and the choice of initial solutions becomes increasingly important. Since finding out a good initial solution itself is a difficult problem, for large sized problems tabu search is implemented in the multi-start mode. In this mode, tabu search is run multiple times, each time with a different initial tour which is widely separated in the solution space from the other initial solutions considered so far. The best tour obtained across all the runs is output as the final solution by the algorithm. The preprocessing algorithms described in Section 2 can be easily tweaked to generate initial tours. In both CE and in PSO, we use the b best tours output by the algorithms (CE and PSO respectively) after the pre-specified number of iterations as b initial tours for multi-start tabu search.

4 Computational Experience

In this section, we first describe the different tabu search implementations that we created to test our preprocessing and initial tour generation schemes. We then describe the experiment design and setting of parameter values, followed by the results of our computational experiments.

4.1 Tabu search implementations

Note that a conventional implementation of tabu search for ATSP instance essentially works on a complete graph irrespective of whether the instance is defined on a complete graph or a sparse graph. In conventional implementations, non-existent arcs in a graph, if any, are represented as infinite cost arcs. Hence even though the neighborhood of a tour is much smaller for an ATSP instance defined on a sparse graph than one defined on a complete graph with the same number of nodes, conventional implementations of tabu search actually search a complete graph in both the cases. Observing this fact, Basu et al. (2013) designed an implementation of tabu search called TS-SAG (tabu search on sparse asymmetric graph) especially for instances defined on sparse asymmetric graphs. TS-SAG uses special data structures to eliminate the need for infinite cost arcs and hence the tabu search actually searches a much smaller neighborhood. This speeds up the TS-SAG algorithm significantly compared to the conventional implementation of tabu search, referred to as TS-CI (tabu search with conventional implementation) on ATSPs defined on sparse graphs.

To test the efficiency of the preprocessing schemes and initial tour generation methods proposed in this paper, we generated five different tabu search based solution procedures for ATSP instances defined on complete graphs. The first of these solution procedure, called RAND-CI, uses the conventional implementation of tabu search with initial tours generated randomly. This solution procedure is used for benchmarking the performance of other solution procedures. The next two solution procedures (CE-CI and PSO-CI respectively) test the efficiency and effectiveness of using initial tours generated by the method based on cross entropy and the method based on particle swarm optimization respectively. Both these solution procedure use the conventional implementation of tabu search. Finally, we test the combined effect of using the initial tours generated by the cross entropy based method and the cross entropy based preprocessing scheme (in the solution procedure CE-SAG) and the initial tours generated by the particle swarm optimization based method and the particle swarm optimization based preprocessing scheme (in the solution procedure PSO-SAG). Both CE-SAG and PSO-SAG use the TS-SAG implementation of tabu search proposed by Basu et al. (2013). Table 1 summarizes the details of these solution procedures that we use for our computational experiments.

Table 1: Details of our solution procedures

Solution Procedure	**Preprocessing Scheme**	**Initial Solution**	**Tabu Search Implementation**
RAND-CI	None	Generated randomly	TS-CI
CE-CI	None	CE	TS-CI
PSO-CI	None	PSO	TS-CI
CE-SAG	CE	CE	TS-SAG
PSO-SAG	PSO	PSO	TS-SAG

As mentioned above comparisons between different solution procedures allow us to comment on the effectiveness of the preprocessing schemes and the initial tour generation methods. Comparison of solution quality of the tours output by procedures RAND-CI, CE-CI and PSO-CI indicates the effectiveness of these methods to generate initial tours for multi-start tabu search. A comparison between procedures pairs (CE-CI, CE-SAG) and (PSO-CI, PSO-SAG) indicates the effectiveness of the preprocessing schemes combined with the use of special tabu search implementation designed for ATSPs defined on sparse graphs. Finally we compare among all five implementations to choose the best one with respect to computational time and average tour quality.

All the implementations were coded in C, were run on a computer with an Intel Quad Core 2.4GHz processor and 3 GB of RAM. The length of the tabu list in all tabu search implementations was fixed at 8 based on the review of literature (Basu, 2012).

4.2 Fixing of parameter values

To set the values of the parameters for our preprocessing schemes, and initial tour generation method we conducted preliminary experiments. For solution procedures based on cross entropy, we conducted preliminary experiments with randomly generated ATSP instances on complete graphs of sizes 250, 500 and 750 nodes. For each problem size we randomly generated 30 problem instances and ran the cross entropy based solution procedure on these instances. The average tour length and computational time for each problem size is reported in the appendix. From the results, we chose the values of k, e, and $iter$ as 50000, $1.5n$ and 20 respectively. We chose the threshold value τ as 1.5 times the average of the costs of arcs in $\mathscr{E}$ weighed by the frequency of their appearance.

For solution procedures based on particle swarm optimization, some of the parameter values such as PSO_{iter}, c_1, c_2, V_{max}, $PSO_{\alpha_{LIF}}$ and PSO_{mp} were taken from the published literature (Bin et al., 2012; Kennedy et al., 2001) as 500, 1, 1, 2, 100 and 0.01 respectively. To set the values of the remaining parameters preliminary experiments were run on 30 randomly generated ATSP instances defined on complete graphs of sizes 250, 500 and 750 nodes. Based on our experiments, for an n city TSP, we fixed the value of swarm size (PSO_{SS}) as $2n$ and the number of leaders (PSO_{lead}) as 15% of PSO_{SS}.

To create the reduced graph, we include all those arcs that appear for more than a certain fraction β of the maximum possible frequency of any arc in leaders across iterations. For example, if the PSO is run for 500 iterations, then we include all those arcs that appear in the leaders for more than $500 \times PSO_{lead} \times \beta$ times. Based on our preliminary experiments we found the value of β as 0.001. To ensure that a tour in the reduced graph has neighboring tour(s), we added all those arcs that appear in the $0.2n$ best tours (PSO_{best}) to the reduced graph. For detailed rationale behind the selection of parameter values, please refer to the appendix.

For tabu search, we used 2-opt neighborhood with tabu list of size 8. We did not include intermediate or long term memory functions, aspiration criteria and strategic oscillation components in our tabu search implementation as out objective is to compare the effect of preprocessing and TS-SAG with a basic tabu search implementation. We have chosen 2-opt neighborhood because of its wide acceptance on TSP in published literature (Basu, 2012). In the same paper, authors reported dominance of fixed tabu tenure with tenure value lying between 5 to 10 therefore we chose tabu tenure as 8.

4.3 Test beds and computational experiments

To compare the performance of each of the five solution procedures mentioned in Table 1, we ran each of these schemes on randomly generated ATSP instances defined on complete graph as well as on benchmark ATSP instances defined on complete graphs. We considered randomly generated instances of sizes 200, 300, 400, 500, and 600 respectively, and for each problem size we generated 30 problem instances. For generating these problem instances, arc costs were chosen randomly as integers in the interval $[1, 1000]$. The benchmark instances consisted of 25 ATSP instances from Johnson et al. (2002) with 100 nodes or more. Note that these instances are also included in TSPLIB (Reinelt, 1991).

We divided our computational experiments into two parts. In the first part we ran tabu search for a fixed number of iterations and in the second part we ran tabu search for a fixed execution time. While the first part allows us to compare the quality of the solutions output by the different solution procedures in general, the second part allows us to compare the quality of the solutions output by the different solution procedures when the solution is required in short time which is the objective of this work.

In the first part, we examined the performance of the five solution procedures when tabu search was allowed to run for 1000 iterations in each implementation. The performance parameters used to compare the procedure in this part for the randomly generated instances were (a) the average of the costs of the best tours output by the procedure on all 30 instances of a given problem size, and (b) the average of the execution times required by the implementation over all 30 instances of a given problem size. For benchmark problem instances, the performance parameters were the cost of the best tour output by the implementation for the instance, and the execution time taken by the implementation.

The second part deals with the performance of the five implementations when the execution time is fixed. We fixed the execution time for different problem sizes in such a way that a sufficient number of tabu search iterations are possible within the specified time limit. We define t_i as the average execution time required by RAND-CI to first encounter the tour it outputs after 1000 iterations on an ATSP instance of size i. Then for ATSPs of size s, we allot an execution time limit of 1000 t_s/t_k seconds, where $k = 500$. The performance measures used in this part are the average of the costs of the tours output by a given implementation on all 30 instances of a particular size for randomly generated problems, and the cost of the tour output by a given implementation on benchmark problem instances.

Computational results (first part): We present the average of the costs of the tours output by the five solution procedures on randomly generated ATSP instances in Table 2. In Table 3 we present the average time required by the five implementations to run the 1000 tabu search iterations on problem instances of a given size. The execution time is broken up into two parts, the time for preprocessing and the time for executing tabu search on the preprocessed instance. The time required to generate the initial tours is included in the time for preprocessing, since the initial tours are generated as a by-product of the preprocessing operation itself.

Table 2: Average of the costs of tours output by the solution procedures at the end of 1000 tabu search iterations on randomly generated instances

	200	**300**	**400**	**500**	**600**
RAND-CI	40520.70	61666.70	85110.10	107101.60	131068.90
CE-CI	19909.20	37726.10	58923.10	80903.00	103893.20
PSO-CI	39143.20	61061.20	83057.5	105113.00	125934.40
CE-SAG	20017.00	38628.30	59630.70	82305.40	106046.30
PSO-SAG	39516.10	70388.00	98272.10	133609.10	189036.00

From Table 2 we see that the tours output by the procedures CE-CI and CE-SAG are significantly better than those output by other procedures. Within procedures involving cross entropy, CE-CI produces slightly better tours than procedure CE-SAG although the tour costs are not significantly different when tested at a significance level of 0.01 (in a paired-t test). Procedures involving PSO, PSO-CI and PSO-SAG, were not able to produce tours of comparable quality as produced by CE after tabu search. Quality of tours output in procedure PSO-CI is marginally better than those produced by RAND-CI, with PSO-SAG producing the worst tour quality among the five procedures experimented.

To compare the effectiveness of initial tours developed by CE and PSO, we compare the output of three procedures: RAND-CI, CE-CI and PSO-CI. For all the three procedures, tabu search was run on a complete graph starting from different initial tours as mentioned in Section 4.1. CE-CI was clearly the best implementation to produce better initial tours for tabu search. To measure the effectiveness of preprocessing schemes in graph reduction, we compared qualities of tours output by CE-CI and CE-SAG for cross entropy and PSO-CI and PSO-SAG for particle swarm optimization. As expected, quality of tours output on a reduced graphs is inferior to the tours output on a complete graph because tabu search is unable to generate all neighboring

Table 3: Average execution time (in seconds) required by the solution procedures to run 1000 tabu search iterations on randomly generated instances

		200	**300**	**400**	**500**	**600**
	Avg. preproc. time	0.00	0.00	0.00	0.00	0.00
RAND-CI	Avg. of time for TS	43.16	127.94	277.31	497.39	821.05
	Avg. of total time	43.16	127.94	277.31	497.39	821.05
	Avg. preproc. time	117.40	264.10	469.20	732.70	1052.50
CE-CI	Avg. of time for TS	42.32	131.77	287.32	533.45	873.23
	Avg. of total time	159.72	395.87	756.52	1266.15	1925.73
	Avg. preproc. time	47.79	128.65	248.56	391.00	600.90
PSO-CI	Avg. of time for TS	42.56	127.04	275.08	490.12	816.35
	Avg. of total time	90.35	255.69	523.64	881.12	1417.25
	Avg. preproc. time	117.40	264.10	469.20	732.70	1052.50
CE-SAG	Avg. of time for TS	3.14	7.44	12.83	19.69	29.18
	Avg. of total time	120.54	271.54	482.03	752.39	1081.68
	Avg. preproc. time	48.44	128.55	248.68	392.05	600.68
PSO-SAG	Avg. of time for TS	27.87	12.64	12.31	13.66	14.03
	Avg. of total time	76.31	141.19	260.99	405.71	614.71

tours. Although the average tour cost in CE-SAG is marginally higher than CE-CI and the difference in average tour costs is well within acceptable limit. In case of PSO, the difference between final tour costs between PSO-CI and PSO-SAG is significant and it increases with problem size. The impact of preprocessing schemes on computational time can be observed from Table 3.

From Table 3 we note that the time required by tabu search in procedure CE-SAG is much less than that required by tabu search in implementation CE-CI. Similar observation is observed while comparing PSO-CI and PSO-SAG. In both cases, difference in computational time increases with increasing problem size. Overall, procedure PSO-SAG takes the least amount of total time among the five implementations followed by implementation CE-SAG.

Table 4 summarizes the quality of tours output by the five solution procedures on the 25 real life benchmark ATSP instances. Since the costs of optimal tours for these problems are very different, we expressed the quality of tours output by the five solution procedures as a multiple of the optimal tour value known for that ATSP instance. Observe that procedures CE-CI and/or CE-SAG produced the least cost tours in 19 out of the 25 instances. The four problems in the rgb class form a notable exception, where procedures RAND-CI/PSO-CI generate the best tours to these problem instances.

Table 4: Costs of tours output by the solution procedures as a multiple of optimal tour cost at the end of 1000 tabu search iterations

		Implementation				
Instance	**Size**	**RAND-CI**	**CE-CI**	**PSO-CI**	**CE-SAG**	**PSO-SAG**
atex8	600	5.330	5.117	5.54	5.12	5.54
big702	702	5.22	5.30	1.37	5.36	3.13
dc112	112	1.01	1.01	1.01	1.01	1.95
dc126	126	1.00	1.00	1.00	1.03	2.12
dc134	134	1.01	1.01	1.02	1.02	4.85
dc176	176	1.02	1.01	1.01	1.03	3.87
dc188	188	1.00	1.00	1.01	1.02	3.38
dc563	563	1.04	1.05	1.04	1.08	5.94
dc849	849	1.05	1.05	1.05	1.05	2.93
dc895	895	1.02	1.02	1.02	1.13	1.03
dc932	932	1.00	1.00	1.00	1.11	1.01
ftv100	100	2.70	2.00	2.48	2.00	2.48
ftv110	110	2.73	2.13	2.25	2.25	2.57
ftv120	120	2.62	2.16	2.16	2.16	2.45
ftv130	130	3.08	2.23	2.23	2.23	3.09
ftv140	140	3.17	2.67	2.69	2.69	3.03
ftv150	150	3.32	2.25	2.43	2.43	3.08
ftv160	160	3.40	2.60	2.67	2.67	3.34
ftv170	170	3.59	2.80	2.80	2.85	3.49
kro124p	100	1.33	1.24	1.24	1.24	1.35
rbg323	323	2.86	3.16	2.88	3.16	3.16
rbg358	358	3.55	4.20	3.52	4.20	4.20
rbg403	403	2.11	2.48	2.05	2.48	2.48
rbg443	443	2.05	2.43	2.02	2.43	2.43
td100_1	100	1.32	1.12	1.30	1.12	1.12

Table 5 presents the execution times required by the procedures on the benchmark problem instances. Between procedures CE-CI and CE-SAG, which provide the best quality tours in most cases, procedure CE-SAG requires much less time, and is hence the preferred procedure. Notice that here too, the difference between the execution times of procedures CE-SAG and CE-CI increases with increasing problem size. Although PSO-SAG takes much less time than the other procedures, it fails to produce good quality tours at end of 1000 TS iterations. Hence, we conducted the second part of our computational experiment to analyze effectiveness of the procedures within a limited time.

Table 5: Execution time (in seconds) required by the procedures to complete 1000 tabu search iterations on benchmark instances

		Implementation				
Instance	**Size**	**RAND-CI**	**CE-CI**	**PSO-CI**	**CE-SAG**	**PSO-SAG**
atex8	600	848.90	854.90	712.50	119.40	10.60
big702	702	1188.60	1238.90	1452.80	61.50	8.80
dc112	112	8.10	7.70	8.40	1.60	7.50
dc126	126	10.40	9.80	8.90	2.00	7.80
dc134	134	12.90	13.80	12.80	2.70	6.90
dc176	176	25.90	22.30	26.50	4.30	14.00
dc188	188	31.60	29.30	28.00	4.70	7.90
dc563	563	675.00	556.00	646.00	31.30	11.90
dc849	849	2139.10	1948.30	2264.30	57.20	7.70
dc895	895	2541.10	2750.30	2317.10	72.50	8.90
dc932	932	2978.90	2932.30	2615.70	75.10	9.00
ftv100	100	6.80	7.10	7.30	2.20	4.50
ftv110	110	9.30	8.60	8.90	2.20	7.70
ftv120	120	14.70	10.70	11.10	4.20	10.50
ftv130	130	12.90	13.40	13.90	6.20	12.60
ftv140	140	15.60	14.60	17.10	4.70	15.90
ftv150	150	20.10	20.10	20.90	7.60	8.40
ftv160	160	24.00	22.90	21.00	6.00	19.50
ftv170	170	29.10	26.50	29.00	10.00	24.50
kro124p	100	6.40	6.10	6.80	4.40	4.60
rbg323	323	138.90	139.80	142.20	12.80	14.60
rbg358	358	319.00	167.50	155.60	15.40	8.60
rbg403	403	208.70	208.10	236.70	18.00	6.30
rbg443	443	351.80	294.10	333.00	21.90	7.50
td100_1	100	6.60	7.20	6.80	4.40	4.70

Computational results (second part): Recall that in the second part of our experiments, we allow each procedure to run for a pre-specified duration, and compare the quality of tours output by the procedure at the end of that duration. Table 6 presents the average of the costs of the tours output by the different procedures over the 30 randomly generated ATSP instances of a given size.

Table 6: Average of the cost of tours output by the procedures at the end of a pre-specified execution time on randomly generated instances

	200	**300**	**400**	**500**	**600**
RAND-CI	39977.60	61282.20	85110.10	107101.60	131616.50
CE-CI	19915.00	37993.60	58871.10	81610.10	105054.20
PSO-CI	39143.20	61061.20	83057.50	105113.00	125934.40
CE-SAG	19819.50	37726.10	59032.90	81428.80	104865.30
PSO-SAG	39516.10	70388.00	98272.10	133609.10	189036.00

The trends in the results from these experiments closely follow their counterparts in the first set. Here too, procedures CE-CI and CE-SAG produce the least cost tours, with CE-SAG producing slightly better quality tours than CE-CI in most cases. We conducted a small set of experiments to see whether the difference in tour quality increases if the time restriction becomes tighter and found that CE-SAG outperformed CE-CI as computational time was further restricted.

For benchmark problem instances, we set the time limit so as to ensure that for each instance tabu search could run from at least three initial tours. The allowable time limits were made proportional to the problem size. The execution times for the 25 instances are given in Table 7.

Table 7: Execution time limits (in seconds) for benchmark problems

Problem	**Time Limit**	**Problem**	**Time Limit**
atex8	2000	ftv120	70
big702	3000	ftv130	80
dc112	50	ftv140	90
dc126	75	ftv150	100
dc134	80	ftv160	110
dc176	120	ftv170	120
dc188	140	kro124p	50
dc563	1500	rbg323	1000
dc849	4500	rbg358	1100
dc895	5500	rbg403	1200
dc932	7000	rbg443	1300
ftv100	50	td100_1	50
ftv110	60		

Table 8 reports the costs of tours output by the procedures as multiples of the optimal solution value for the corresponding problem. We see that procedures CE-CI and/or CE-SAG produced the best tours in 22 of the 25 benchmark instances.

Table 8: Costs of tours output by the procedures as a multiple of optimal solution value at the end of the pre-specified execution time

		Implementation				
Instance	**Size**	**RAND-CI**	**CE-CI**	**PSO-CI**	**CE-SAG**	**PSO-SAG**
atex8	600	5.45	5.19	3.17	4.99	5.42
big702	702	12.65	4.91	1.89	5.19	2.63
dc112	112	1.01	1.01	1.97	1.01	1.96
dc126	126	1.00	1.00	1.00	1.02	1.00
dc134	134	1.01	1.01	4.92	1.02	4.85
dc176	176	1.02	1.01	4.15	1.03	4.15
dc188	188	1.00	1.01	3.66	1.02	3.38
dc563	563	1.05	1.08	4.60	1.05	5.94
dc849	849	1.05	1.05	5.09	1.05	5.78
dc895	895	1.03	1.03	1.86	1.13	2.48
dc932	932	1.00	1.11	1.00	1.01	1.00
ftv100	100	2.62	1.88	3.71	1.88	2.93
ftv110	110	2.73	1.90	10.60	1.90	10.49
ftv120	120	2.63	2.12	10.87	2.12	10.87
ftv130	130	3.01	2.28	11.13	2.28	11.13
ftv140	140	3.17	2.50	10.83	2.40	11.58
ftv150	150	3.31	2.57	11.34	2.57	11.98
ftv160	160	3.40	2.74	11.89	2.64	11.89
ftv170	170	3.59	2.81	12.25	2.80	11.45
kro124p	100	1.33	1.24	1.35	1.24	1.35
rbg323	323	3.61	2.19	49.30	2.20	49.81
rbg358	358	859.84	2.72	64.76	2.78	70.87
rbg403	403	405.68	1.69	34.04	1.74	47.48
rbg443	443	367.65	1.74	35.31	1.79	44.82
td100_1	100	1.32	1.13	1.36	1.12	1.30

To summarize, we observe that the preprocessing schemes and the initial tour generation method described in this paper combined with the TS-SAG implementation of tabu search produce results for moderate sized ATSP instances which are superior to conventional tabu search implementations for this problem.

Next we compare the efficiency of CE-SAG with two well known heuristics on ATSP: KP heuristic by Kanellakis and Papadimitriou (1980) and repeated local search heuristic by Helsgaun (2000). Selection of these two heuristics is motivated by the following findings from the paper by Johnson et al. (2002). In this paper, four heuristics were chosen based on their performances on ATSP: Patch heuristic (Karp and Steel, 1985) (PATCH), KP heuristic by (Kanellakis and Papadimitriou, 1980) (KP), Helsgaun heuristic (Helsgaun, 2000) (HG) and cycle cover heuristic by Zhang (2000) (ZH). Although TS-SAG takes less computational time compared to all four heuristics on most of the benchmark instances, PATCH and ZH obtain significantly better results than CE-SAG in most of the benchmark instances. Therefore, we do not report the results obtained on these instances, but report results on some 'dc' instances in Table 9 on which CE-SAG produces results of comparable quality with those output by KP and HG with significant reduction in computational time. The time in seconds for KP and HG are taken from results reported in a paper by Johnson et al. (2002) with experiments conducted using 196 MHz MIPS R10000 processors. The difference in computational time between CE-SAG and other heuristics increases with problem size. Like on 'dc112' instance, heuristics KP and HG take 14 and 3 seconds more respectively than CE-SAG whereas on 'dc932' instance, heuristics KP and HG take 44 and 72300 seconds more respectively than CE-SAG.

Table 9: Comparison of solution quality and computational time with standard heuristics

	% Excess over Optimal			**Running Time (in secs)**			
Instance	**CE-SAG**	**KP**	**HG**	**CE-SAG**	**KP**	**HG**	**OPT**
dc112	1.25	0.39	0.28	1.6	15.47	4.5	76.2
dc126	2.65	0.65	0.54	2	22.69	5.5	48.3
dc134	1.59	0.57	0.02	2.7	13.43	6.5	63.7
dc176	2.53	0.67	0.49	4.3	20.48	105	195.1
dc188	1.58	0.59	0.13	4.7	12.98	28.5	122.8
dc563	8.29	0.79	0.12	31.3	111.95	2231.5	1449.7
dc849	4.59	0.62	0.23	57.2	114.8	2180	378.3
dc895	12.97	0.6	0.25	72.5	144.43	22077	35926.1
dc932	10.64	0.26	0.26	75.1	119.17	72373	14722.4

5 Summary

In this paper, we develop a tabu search implementation to solve asymmetric traveling salesman problems (ATSPs). In our implementation, we first develop two different preprocessing algorithms based on cross entropy and particle swarm optimization to reduce the density of the graph describing the problem instance, and to generate good starting tours for tabu search to operate in a multi-start mode. We then use a tabu search implementation specially designed to solve ATSPs defined on sparse graphs.

We created five solution procedures, RAND-CI without any of the enhancements suggested in the paper, CE-CI and CE-SAG in which cross entropy is used to generate initial tours and to preprocess, PSO-CI and PSO-SAG to create initial tours and to preprocess using particle swarm optimization. The special tabu search implementation designed for sparse graphs from Basu et al. (2013) could only be used in CE-SAG and in PSO-SAG. We compared the solution procedures based on extensive computational experiments using randomly generated ATSP instances and benchmark problem instances. We performed two types of experiments, one in which we fixed the number of tabu search iterations, and another in which we fixed the total execution time. Our experiments indicate that the best tabu search implementation for medium sized ATSPs is the one in which both the preprocessing and the initial tour generation are done based on the cross entropy method, and on the resulting sparse graph the TS-SAG tabu search implementation (see Basu et al., 2013) is applied.

The work in this paper contributes to the existing literature of metaheuristic applications on combinatorial optimization problems like ATSP. Although we have considered ATSP as our problem context in this paper, the idea of combining preprocessing methodology with metaheuristic implementation is novel and can be extended to other combinatorial optimization problems with minor modification. Combining preprocessing with metaheuristics enabled us to speed up standard tabu search implementation significantly and our results are comparable to some of the best known heuristics in ATSP literature. As emphasized in the introduction, objective of our work is to address large sized ATSPs which requires significant reduction in computational time without much deterioration in quality of the final tour output. CE-SAG, conceptually simple and easy to understand, shows significant improvement in computational time on randomly generated problem instances till 600 nodes and real life benchmark instances. We have also used these preprocessing algorithms to generate diversified good initial solutions suitable for multi-start tabu search. Relevance of multi-start local search approach increases with large solution space present in large problem sizes.

The solution procedures in this paper are restricted to developing preprocessing algorithms using cross entropy and particle swarm optimization. Although we are able to show some encouraging results in terms of reduction of computational time, our experiments can be extended by considering other well known construction heuristics (Glover et al., 2001) or population heuristics (Choi et al., 2003) for graph preprocessing and initial tour generation. Also in this paper, we intended to see the effect of preprocessing mechanisms on a basic metaheuristic framework. Our effort can be extended by making suitable changes within the components of the metaheuristic itself, e.g. random tabu tenure, aspiration criteria etc., which may lead to further improvement in solution quality. In future, we plan to test the effect of similar preprocessing mechanisms on other combinatorial optimization problem instances.

References

Applegate D, Bixby R, Chvatal V and Cook W (2006). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

Basu S (2012). Tabu search implementation on traveling salesman problems and its variations: A literature survey. *American Journal of Operations Research* **2**: 163–173. W.P. No. 2008-10-01.

Basu S (2012). Neighborhood reduction strategy for tabu search implementation in asymmetric traveling salesman problem. *OPSEARCH* **49**: 400–412.

Basu S, Gajulapalli R and Ghosh D (2013). A fast tabu search implementation for large asymmetric traveling salesman problems defined on sparse graphs. *OPSEARCH* **50**: 75–88.

Bin W, Qinke P, Jing Z and Xiao C (2012). A binary particle swarm optimisation algorithm inspired by multi-level organisational learning behavior. *European Journal of Operational Research* **219**: 224–233.

Boer P, Kroese D, Mannor S and Rubinstein R (2005). A tutorial on the cross-entropy method. *Annals of Operations Research* **134**: 19–67.

Chepuri K and Homem-de Mello T (2005). Solving the vehicle routing problem with stochastic demands using the cross entropy method. *Annals of Operations Research* **134**: 193–181.

Choi I-C, Kim S-I and Kim H-S (2003). A genetic algorithm with a mixed region search for the asymmetric traveling salesman problem. *Computers & Operations Research* **30**: 773–786.

Cirassela J, Johnson D, McGeoch L and Zhang W (2001). The asymmetric traveling salesman problem: algorithms, instance generators, and tests. In:

Buchsbaum A and Snoeyink J (eds). *Algorithm Engineering and Experimentation*. Third International Workshop, ALENEX 2001, Lecture Notes in Computer Science **2153**, pp 32–59. Springer-Verlag.

Fiechter C-N (1994). A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics* **51**: 243–267.

Fischetti M, Lodi A and Toth P (2002). Exact methods for asymmetric traveling salesman problem. In: Gutin G and Punnen P (eds). *The Traveling Salesman Problem and Its Variations* **4**, pp 169–206. Kluwer Academic Publisher: London.

Glover F (1989). *Tabu Search-Part I. INFORMS Journal of Computing* **1**: 190–206.

Glover F (1990). *Tabu Search-Part II. INFORMS Journal of Computing* **2**: 4–32.

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

Glover F, Gutin G, Yeo A and Zverovitch A (2001). Construction heuristics for the asymmetric TSP. *European Journal of Operational Research* **129**: 555–568.

Goossens J and Baruah S (2001). Multiprocessor preprocessing algorithms for uniprocessor on-line scheduling. In: *The 21th International Conference on Distributed Computing Systems*.

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

Held M and Karp R (1971). The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming* **1**: 6–25.

Helsgaun K (2000) An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research* **12**: 106–130.

Johnson D, Gutin G, McGeoch L, Yeo A, Zhang W and Zverovich A (2002). Experimental analysis of heuristics for the ATSP. In: Gutin G and Punnen P (eds). *The Traveling Salesman Problem and Its Variations* **10**, pp 445–488. Kluwer Academic Publishers: London.

Kanellakis P C and Papadimitriou C H (1980) Local search for the asymmetric traveling salesman problems. *Operations Research* **28**: 1066–1099.

Karp R (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations*, pp 85–103. Plenum Press.

Karp R and Steel J M (1985). Probabilistic analysis of heuristics. In A H G Rinnooy Kan, E L Lawler, J K Lenstra and D B Shmoys, editors. *The traveling salesman problem: A guided tour of combinatorial optimization* Wiley, Chichester.

Kennedy, J. and Eberhart, R (1995). Particle swarm optimization *Proceedings of IEEE International Conference on Neural Networks* **1-6**: 1942–1948.

Kennedy J, Eberhart R C, Chvatal V and Shi Y (2001). *Swarm Intelligence.* Morgan Kaufman.

Kennedy, J. and Eberhart, R (1997). A discrete binary version of the particle swarm algorithm *Proceedings of IEEE International Conference on Systems, Man, and Cybernatics, Computational Cybernatics and Simulation* **Cat No. 97CH36088-5**.

Khumawala B (1975). An efficient branch and bound algorithm for the warehouse location problem. *Management Science* **18**: B718–B731.

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

Laporte G (2010). A concise guide to the traveling salesman problem *Journal of Operational Research Society* **61**: 35–40.

Lin S and Kernighan B W (1973) An effective heuristic algorithm for the traveling salesman problem. *Operations Research* **21**: 972–989.

Reinelt G (1991). TSPLIB – A traveling salesman problem library. *ORSA Journal on Computing* **3**: 376–384.

Roberti R and Toth P (2012). Models and algorithms for the Asymmetric Traveling Salesman Problem: an experimental comparison. *European Journal of Transportation and Logistics* **1**: 113–133.

Rubinstein R (1997). Optimization of computer simulation models with rare events. *European Journal of Operational Research* **99**: 89–112.

Rubinstein R (1999). The simulated entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability* **2**: 127–190.

Rubinstein R (2001). Combinatorial Optimization, Cross-Entropy, Ants and Rare Events. In: Uryasev S and Pardalos P M (eds). *Stochastic optimization: algorithms and applications*, pp 445–488. Kluwer Academic Publishers: London.

Tang L, Liu, J, Rong, A and Yang, Z (2000). A multiple traveling salesman problem model for hot rolling scheduling in Shanghai Boston Iron and Steel Complex. *European Journal of Operational Research* **124**: 267–282.

Toth P and Vigo D (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing* **15**: 333–346.

Zhang W (2000). Truncated branch-and-bound: Case study on the asymmetric TSP. *n Proceedings of 17th National Conference on Artificial Intelligence AAAI 2000*: 930–935. Austin, TX.

Appendix

1 Tabu search implementation for sparse asymmetric traveling salesman problem

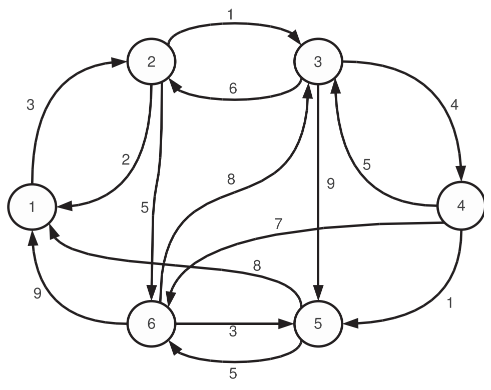
We developed a modified tabu search implementation suitable for sparse asymmetric travels salesman problem instances. This implementation exploits the sparse nature of the underlying graph to reduce computational time. Two main data structures used in any tabu search implementation for TSPs are structures to store the graph describing the problem and structures to store information about tours. In this section we first describe the data structures in our implementation to store graphs in Subsection 1.1. We then describe the data structures to store tours in Subsection 1.2. We finally describe how we use these data structures to increase the speed of tabu search iterations in Subsection 1.3. In this section, we will assume that the graph defining the ATSP has n nodes, and each node is directly connected to k other nodes in the graph on average.

1.1 Data structures to store graphs

Complete graphs are predominantly used in tabu search literature and therefore the most efficient 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 sufficient 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 inefficient. Some papers elaborate on different data structures used to store sparse matrices, of which the compressed row format is especially effective. 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.



tail	head	cost
1	2	3
2	1	2
2	3	1
2	6	5
3	2	6
3	4	4
3	5	9
4	3	5
4	5	1
4	6	7
5	1	8
5	6	5
6	1	9
6	3	8
6	5	3

`arcs`

1	2	5	8	11	13

`arc_position_ptr`

Fig. 1: A graph for an ATSP instance and its representation in our implementation

With our data structures, the time required to search whether a particular node is directly connected to another node is $O(\log(k))$ on average, and the time required to list all neighbors of a node is $O(k)$ on average. In an adjacency matrix representation, the first operation is constant time, while the second requires $O(n)$ time.

1.2 Data structures to store tours

In conventional implementation, tours are either stored as a permutation of nodes in V or as a linked list of arcs. Storing tours in the latter format has obvious advantages when one is dealing with symmetric TSPs on complete graphs; in such cases, converting a tour to neighboring tours is a constant time operation of modifying four pointers. However, this is not true when one is dealing with ATSPs, since a 2-opt move in an ATSP requires the sequence of intermediate nodes to be reversed. In addition, when dealing with ATSPs defined on sparse graphs, the existence of a chain of nodes in one direction is no guarantee that the chain will remain feasible when the arc directions 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 $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 1$ in the graph in Figure 1. Assume that the tabu list is $\{(2, 6),\ (3, 2),\ (3, 5),\ (4, 3)\}$. Figure 2 represents the tour

tail	head	arc cost	rever-sible?	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`

Fig. 2: Representation of a tour in our implementation

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 in the fourth row of `tour-arcs`, columns 5 and 6 have a '*' since the arc (5,4) does not exist in the graph, as noted in column 4 of the array. □

1.3 Tabu Search for Sparse Asymmetric Graph (TS-SAG)

We make tabu search efficient for problems defined on sparse graphs by devising a method to quickly find legitimate candidate arcs for 2-opt moves given a tour and a tabu list, by utilizing the sparseness of the graph defining the ATSP. Given a graph $G = (V, A, C)$, a tour τ and a tabu list L, Algorithm 1 presents our method of finding the best neighboring tour which does not include any arc present in L. This algorithm therefore defines the main part of a tabu search iteration for our implementation. For notational convenience, a tour τ is a vector of arcs $(a_1, a_2, \ldots, a_n)$ where the head of arc a_i is the tail of arc a_{i+1} for $i = 1, \ldots, n-1$, and the head of arc a_n is the tail of arc a_1, and $[\tau \circledast (a_i, a_j)]$ is the tour obtained from tour τ through a 2-opt operation involving the deletion of arcs a_i and a_j.

Algorithm 1 Finding the best neighbor of a tour in our implementation

```
Input:  $G = (V, A, C)$, A tour $\tau$, tabu list $L$
Output: The best neighbor $\tau_{best}$ of $\tau$
 1: /* INITIALIZATION */
 2: set $b \leftarrow \infty$
 3: set $\tau' \leftarrow \phi$
 4: /* ITERATION */
 5: for all $a_i = (p, q) \in \tau$ do
 6:     set $flag(a_j) \leftarrow 0$ for each $a_j \in \tau$
 7:     for all $a_j = (r, s) \in \tau$ such that $(s, r) \in A \setminus L$ do
 8:         set $flag(a_{j+1}) \leftarrow 1$
 9:     end for
10:     for all $a_j = (r, s) \in \tau$ such that $flag(a_{j-1}) = 1$ and $(p, r) \in A \setminus L$ do
11:         set $flag(a_j) \leftarrow 2$
12:     end for
13:     for all $a_j = (r, s) \in \tau$ such that $flag(a_j) = 2$ and $(q, s) \in A \setminus L$ do
14:         set $flag(a_j) \leftarrow 3$
15:     end for
16:     for all $a_j (\neq a_i) \in \tau$ such that $a_i$ and $a_j$ are not adjacent
        and $flag(a_j) = 3$ do
17:         if $c([\tau \circledast (a_i, a_j)]) < b$ then
18:             set $a_1^* \leftarrow a_i$, $a_2^* \leftarrow a_j$, and $b \leftarrow c([\tau \circledast (a_1, a_2)])$
19:         end if
20:     end for
21: end for
22: set $\tau_{best} \leftarrow [\tau \circledast (a_1^*, a_2^*)]$
23: return $\tau_{best}$
```

Our implementation is fast because we are able to, without explicit computation, eliminate infeasible 2-opt neighbors of a given tour. We do this by manipulating a flag value (see steps 6 through 15 in Algorithm 1). To understand how the flag value is manipulated, note that two arcs a_i and a_j will yield a feasible 2-opt neighbor only if the following three conditions are met.

1. For each arc (p, q) between a_i and a_j in the tour, the arc (q, p) must exist in the graph, and not be in the tabu list;
2. there must exist an arc from the tail of a_i to the tail of a_j in the graph and it should not be in the tabu list; and
3. there must exist an arc from the head of a_i to the head of a_j in the graph and it should not be in the tabu list.

Now suppose, without loss of generality, that we want to find those arcs with which arc a_1 can pair to yield feasible 2-opt neighbors. We start by initializing the flag value for each arc to 0, and incrementing the flag value for each arc by 1 for each one of the three conditions that it satisfies. Step 6 in Algorithm 1 performs this operation. Steps 7 through 9 implement condition 1, steps 10 through 12 implement condition 2, and steps 13 through 15 implement condition 3. At the end of the incrementing, only those arcs which have a flag value of 3 satisfy all the conditions to be eligible to participate in a 2-opt operation with a_1. Of course a_2 and a_n, being adjacent to a_1 cannot participate in a 2-opt operation with it and can be ignored. Also, if we set flag values starting from a_3 and going up to to a_{n-1}, and if we find an arc a_i which does not satisfy condition 1, then we can conclude that none of the arcs from a_{i+1} through a_{n-1} will satisfy condition 1. So we do not need to check those arcs for conditions 2 and 3. This observation speeds up the update of flag values significantly. Through this process, Algorithm 1 identifies the same number of neighboring tours as done in a conventional 2-opt move. While implementing in a sparse asymmetric graph, our algorithm identifies the pairs of arcs to be deleted in a 2-opt move to yield a feasible neighboring tour before actually executing it.

We now compute the complexity of Algorithm 1. Consider the flag value computation to identify all arcs which can participate in a 2-opt operation with a particular arc a_i in the tour. Given our tour data structure, checking condition 1 for all arcs in a tour requires $O(n)$ time. Finding the list of nodes that are directly connected to a node in the graph requires $O(k)$ time on average. Checking if an arc connecting two nodes is in the tabu list requires $O(\log(|L|))$ time. So performing steps 2 and 3 require $O(k \log(|L|))$ time on average. Therefore the flag value computation requires $\max\{O(n), O(k \log(|L|))\}$ time on average. If k and $|L|$ are small compared to n, as is the case in 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 $O(n^3)$.

2 Preliminary experiments for parameter settings

In this section, results of our preliminary experiments were reported to fix parameter values for the final experiment. We conducted preliminary experiments in two stages. In the first stage, we ran preprocessing schemes to check solution values and computational time. If the first stage results after preprocessing were not sufficient to fix the parameter values, we ran tabu search on reduced graph. Using final tour values and computational time, we decided on parameter values.

2.1 Cross Entropy Intensification

For the basic cross entropy method, we require to fix the following parameter values:

- Number of random tours to form set k = $|\mathscr{K}|$
- Number of elite tours to form set e = $|\mathscr{E}|$
- Maximum number of cross entropy iterations ($ceiter_{max}$)

In the first stage of experiments, we implement cross entropy on a complete graph data set. Problem characteristics and ranges for parameter values tested are as follows:

Preliminary Experiments - Stage I

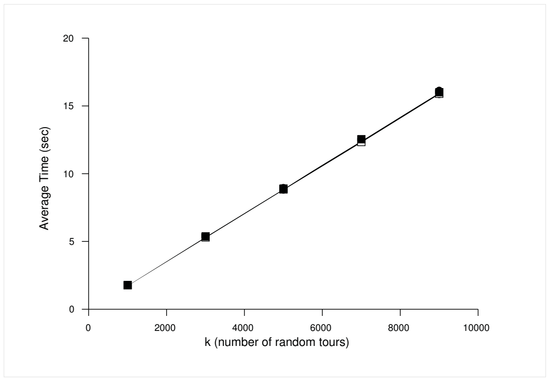
size: 100, 250, 500;

k: 1000, 3000, 5000, 7000, 9000;

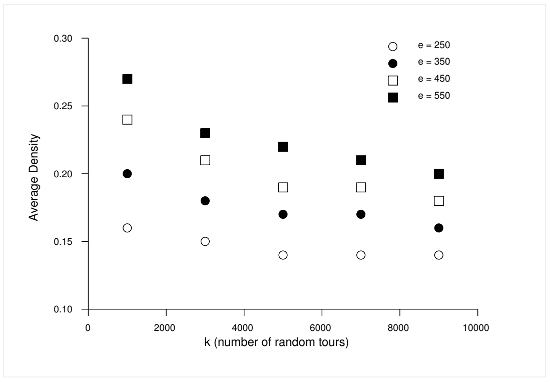
e: n, $n+100$, $n+200$, $n+300$ for a n node TSP;

ceiter$_{max}$**:** 0 to 49;

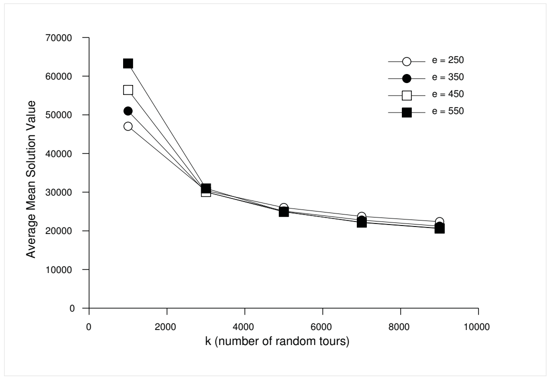
We took ten problem instances in each problem size category. We report the results for one problem size as similar results were obtained for other problem sizes. We show the results from Figures 3a to 3c for a 250 node ATSP.



(a) Change in average time taken in 250 node TSP with change in k value

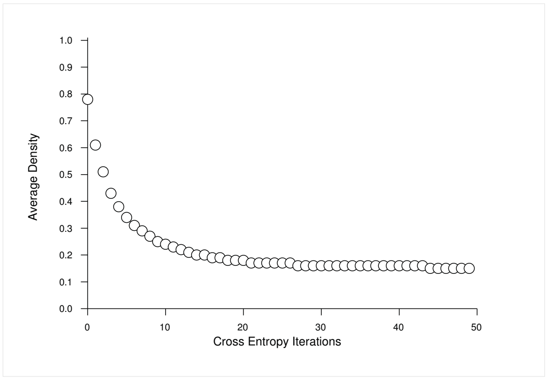


(b) Change in average density on 250 node TSP with changes in k and e values

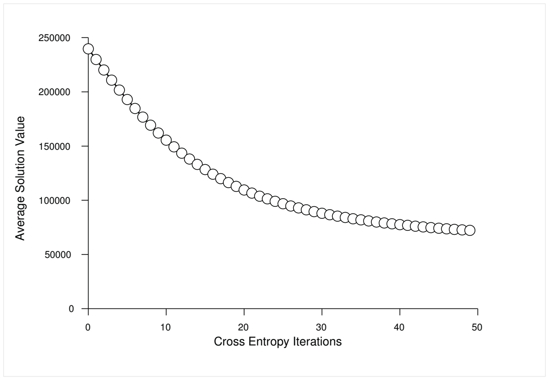


(c) Change in average tour distance on 250 node TSP with changes in k and e values

Fig. 3: Impact of changes in values of k and e



(a) Change in average density with CE iterations on 500 node TSP



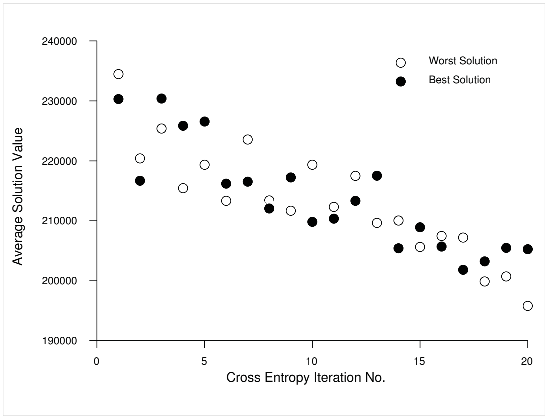
(b) Change in average tour quality with CE iterations on 500 node TSP

Fig. 4: Impact of changes in CE iterations

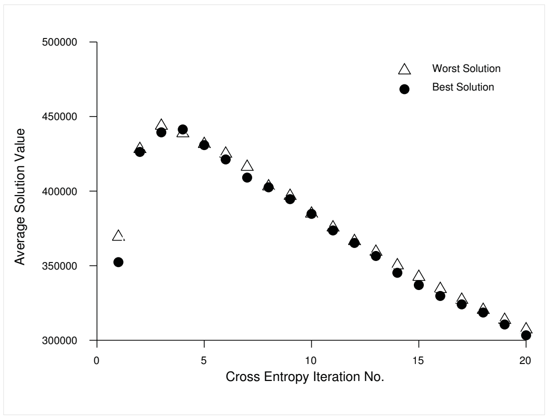
In Figure 3a, we observe a linear increase in total time with increase in k values. There is no effect of the change in value of e on total time taken for cross entropy. From Figure 3b, we observe that increase in values of k leads to a decrease in average graph density whereas increase in number of elite tours chosen (e) obviously increases graph density. Average tour quality gets better with increase in values of both k and e as shown in Figure 3c. From these three figures, we had fixed the value of k as 5000 and e as $1.5n$ to obtain a reasonably good initial tour for tabu search without taking significant amount of computational time. We also intended to keep the density of the reduced graph at least in the range of $15-25\%$ to allow tabu search to find some neighbouring tours.

After fixing the values of k and e, we ran experiments to decide on number of CE iterations. Results of this experiment on a 500 node ATSP is presented in Figures 4a and in 4b. In both the graphs, average tour diastase and graph density reduce with increase in number of iterations. Based on these observations, We set number of cross entropy iterations as 20 to get a balance between density level and tour quality.

Through this experiments, although we were able to find initial tours of good quality for tabu search, it might lead to a solution domain where tabu search were unable to improve much on that. To test this argument, we ran tabu search using the best 10 tours and worst 10 tours from the elite list and presented the average tour quality in Figures 5a and in 5b. The effect of initial tour quality on the final tour output after tabu search is analyzed in Figures 5a and in 5b. We chose the best 10 tour and worst 10 tours to see their impact on final tour distance. From the figures, we conclude that starting tabu search from good initial tours has a definite impact



(a) Change in average tour quality after TS on complete graph with different initial tours



(b) Change in average tour quality after TS on reduced graph with different initial tours

Fig. 5: Impact of initial tours on TS performance

on final tour value when implemented on reduced graph.

2.1.1 Cross Entropy with Elite Arc Criteria by Toth and Vigo

We have used the concept of THRESHOLD DISTANCE from the paper by Toth and Vigo, which is the average length of all arcs present in the elite tour set. To accommodate more number of arcs into the reduced graph, we increase the THRESHOLD DISTANCE by multiplying it to a parameter named MULTIPLIER. To look into the influence of this parameter, we took 10 instances of 500 node TSP and constructed reduced graphs after cross entropy for different values of MULTIPLIER. Tabu search was performed on these reduced graphs to find the variation in final tour quality obtained. We illustrate the results in Figure 6 with MULTIPLIER value in X-axis, average tour distance after tabu search and average graph density after CE in primary and secondary Y axes respectively. Based on the results, we chose MULTIPLIER value as 1.5 to keep a balance between average graph density and average tour quality.

2.2 Particle Swarm Optimization

From the set of parameters values required for PSO, we fixed values of following parameters from the literature:

- PSO_{iter} : 500
- $c_1 = 1$

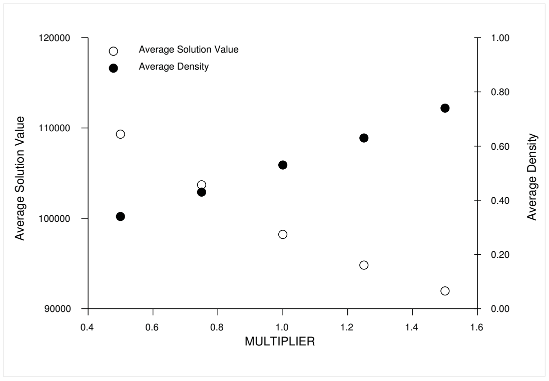


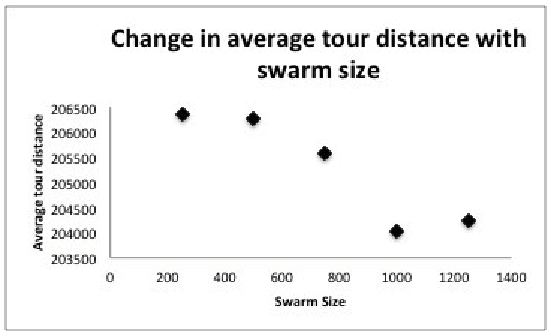
Fig. 6: Change in average tour quality after tabu search and average density after CE for different MULTIPLIER values

- $c_2 = 1$
- $PSO_{\alpha_{LIF}} = 100$
- $PSO_{mp} = 0.01$
- $PSO_{V_{max}} = 2$

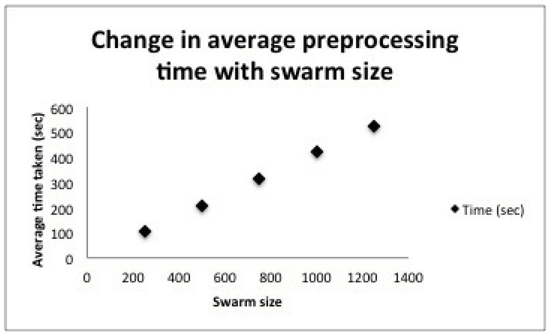
For remaining parameter values, we conducted a set of preliminary experiments with a problem size of 500 node TSP with 30 problem instances. To determine swarm size (PSO_{SS}), we ran experiments to see the change in average tour distance and preprocessing time. We consider swarm size as a function of problem size and tested for swarm size values from n to $2.5n$ with a problem size of n. The results are illustrated in Figures 7a and 7b. We chose swarm size as $2n$ because it provided with the best tour quality within reasonable computational time. We also investigated with larger swarm sizes but the solution quality had improved marginally with linear increase in preprocessing time.

We chose a set of leader tours from the swarm size to influence follower tours in the next iteration. Number of leaders (PSO_{lead}) was considered as a function of swarm size. We took PSO_{lead} as a fraction of PSO_{SS} by considering fraction values 0.10, 0.15, 0.20 and 0.25. The results are reported in Figures 8a and 8b. We found that taking PSO_{lead} as 15% of PSO_{SS} yielded better results in terms of average tour quality and computational time for most of the cases. Hence we fixed number of leaders as 15% of swarm size.

To create a reduced graph $G(V, A')$, we had included the arcs (A'_L) present among the leaders with a frequency more than the threshold frequency defined for the leaders ($PSO_{thresh-lead}$). To further increase the density of the graph, set of best tours (TB

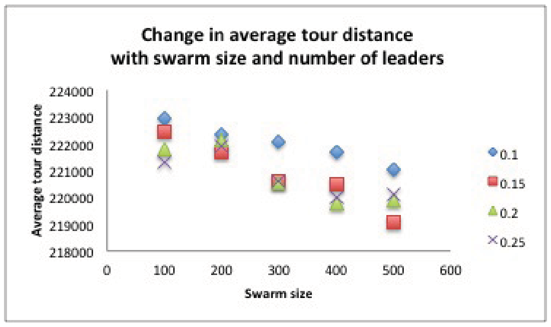


(a) Change in average tour quality with changes in swarm size for a 500 node TSP

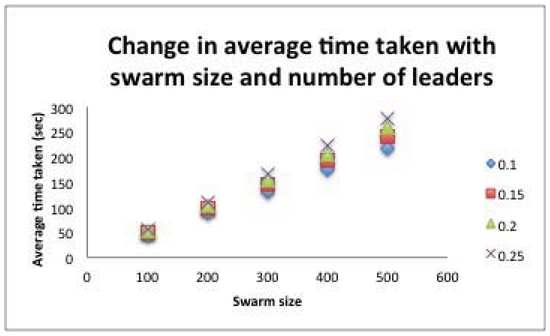


(b) Change in average preprocessing time with changes in swarm size for a 500 node TSP

Fig. 7: Impact of swarm size on solution quality and preprocessing time



(a) Change in average tour quality with changes in selection of number of leaders



(b) Change in average preprocessing time with changes in selection of number of leaders

Fig. 8: Selection of number of leaders as a fraction of swarm size

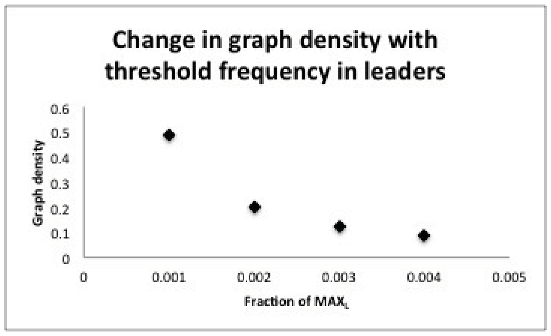


Fig. 9: Change in graph density with change in $PSO_{lead-thresh}$

with $|TB| = PSO_{best}$) are tracked throughout iterations to add the corresponding arcs from arc set A_T. For a formal definition, we use the expressions below:

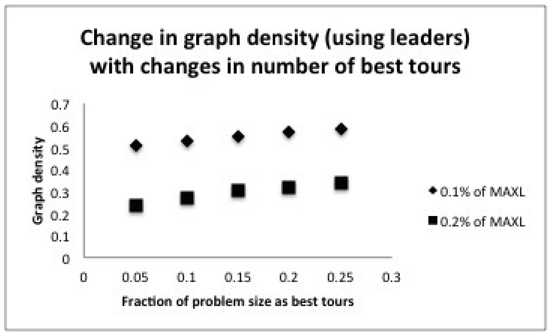
$$A' \leftarrow A'_L \cup A_T \text{ with } A'_L = \{a : n^a_L \geq T_L\} \text{ and } A_T = \{a : a \in TB\}$$

A'_L is an arc set considering the arcs with frequency in leaders (n^a_L) is at least equal to the threshold frequency $PSO_{thresh-lead}$. Additionally, we added the arcs (from set A_T) present in the best TB tours found across PSO iterations to increase graph density.

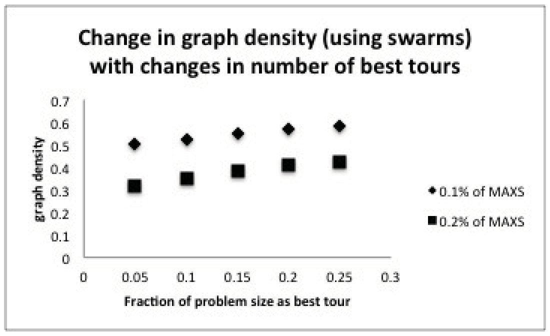
To define $PSO_{thresh-lead}$, we took it as a fraction of maximum possible frequency of an arc in leaders (MAX_L). For different fraction values of $MAX_L = (PSO_{lead} \times PSO_{iter})$, the results were shown in Figure 9. Selection of threshold values has an impact on the density of the reduced graph and hence we chose to report accordingly. While reporting the graph density, we did not include the arcs present in A_T to find the impact of threshold values independently.

From the previous set of preliminary experiments involving cross entropy, we found that the required graph density should be in the range of $50-70\%$ to run tabu search effectively. Apart from fraction values of 0.001 and 0.002 of MAX_L, graph densities obtained were too low. Hence we did the next set of experiments with these two fractional values along with adding the arcs presents in A_T. To find a suitable number of tours in A_T, we experimented with values of $0.05n$, $0.1n$, $0.15n$, $0.2n$ and $0.25n$ with n as problem size. The results were illustrated in Figures 10a and 10b.

Based on the results shown, we chose threshold frequency as 0.1% of MAX_L along with $|A_T| = 0.2n$ for our final experiments. It gave us the required density level in between 0.5 to 0.6. Increasing the number of best tours may increase the density further but only in expense of more computational time. Also we have used tours in leaders to choose arcs for reduced graph as the preliminary experiment shows a



(a) Change in graph density with change in T_L and A_T



(b) Change in graph density with change in T_S and A_T

Fig. 10: Selection of arcs based on threshold frequency and number of best tours

slightly higher density level than selecting from swarms. To summarize the selection of parameters values (expression), please see the following table for a TSP with n nodes:

Tab. 1: Final parameter values (expressions) for preprocessing with PSO

Parameter	**Values (expressions)**
PSO_{iter}	500
c_1	1
c_2	1
PSO_{VMAX}	2
$PSO_{\alpha_{LIF}}$	100
PSO_{mp}	0.01
$PSO_{V_{max}}$	2
PSO_{SS}	$2n$
PSO_{lead}	$0.15.PSO_{SS}$
$PSO_{thresh-lead}$	$0.001.MAX_L$
A_T	$0.2n$

3 Setting of time limits for experiments with fixed execution time

The second part of our computational experiment deals with the performance of tabu search implementations when the execution time was fixed. We fixed the execution time for different problem sizes ensuring that a sufficient number of tabu search iterations are possible from at least two initial tours within the specified time limit.

We assigned the execution time of 1000 seconds for a 500 node TSP seeing the preprocessing and tabu search time it requires in our first set of experiment (with fixed number of iterations). We defined t_k as the execution time that implementation A required to first encounter the tour it output after 1000 iterations on a ATSP instance of size k. Then for ATSPs of size s, we allot an execution time limit of 1000 t_s/t_{500} seconds (T_1). We also see the minimum time it requires to run tabu search from at least two initial tours (T_2). Then we chose the maximum of T_1 and T_2. Final time limits set for different problem sizes are as follows:

Tab. 2: Execution time limits (in seconds) for different problem sizes

Problem size	**Execution time (secs)**
200	150
300	400
400	700
500	1000
600	1500