COMPLEXITY INTERNATIONAL

Volume 11

Tracking moving targets and the non-stationary traveling salesman problem

Q. Jiang, R. Sarker and H. Abbass

School of Information Technology and Electrical Engineering
University of New South Wales at ADFA
Northcott Drive, Canberra 2600, Australia
Email: {ruhul, abbass}@cs.adfa.edu.au

Abstract

The Traveling Salesman Problem (TSP) has been studied extensively in the literature with the assumption that all cities to be visited are stationary. In this paper, we investigate a non-stationary version of TSP (NTSP) where all cities (objects/targets) are moving at known velocities. This problem is motivated by many real life problems in security and defence. We propose a genetic algorithm based solution approach for NTSP and analyse the solutions obtained.

1. Introduction

The Traveling Salesman Problem (TSP) is one of the best known and extensively studied optimization problems in the literature. It has attracted the attention of many researchers over the last half-a-century because of its simple problem description but difficulty in obtaining the optimal solutions efficiently. The problem is: a salesman, starting from her base, intends to visit each of several cities exactly once and return to the base ensuring minimum total traveling distance (or cost).

The classical TSP has been studied extensively, and many algorithms have been proposed over the last half-a-century. However, many real-world TSP type problems are dynamic in nature. Recently, Zhou et al (2003) studied the dynamic or non-stationary TSP (NTSP) which may encompass one or more additional features such as (i) the number of cities may change with time: some new cities may appear in the tour and some old ones may disappear; and/or (ii) the city locations (or cost matrix) may change with time. In this paper, we discuss a non-stationary version of TSP where the number of cities is fixed but the cost matrix changes continuously.

In the following section, we define NTSP very briefly. The following two sections discuss the solution approaches for classical TSP and a genetic algorithm based solution approach for

NTSP. Experimental setup and analysis of results are then presented and conclusions are drawn.

2. Non-stationary TSP

Most previous works on TSP have assumed that the number of cities is fixed and the cities (objects/targets) to be visited are stationary. However, there are practical TSP scenarios where the targets to be visited are themselves in motion (Helvig et al, 2003). For examples, (i) re-supplying patrolling boats by a supply ship, (ii) intercepting a number of mobile ground units by an aircraft, (iii) intercepting a number of non-stationary ships by an aircraft and (iv) supplying hazardous materials to a number of mobile units by a robot. In this paper, we consider a NTSP where the targets move with constant velocities on a known two-dimensional surface. We define the NTSP as follows:

> Given a set of targets $S = \{s_1, \ldots, s_n\}$, each target s_i is moving in a direction d_i at constant velocity v_i from an initial position p_i, and given a pursuer starting from the origin (say O) at constant velocity v_p, find the shortest/fastest tour starting (and ending) at the origin, such that the pursuer intercepts with all targets.

To describe an NTSP, we have considered one pursuer starting from the origin O and three targets s_1, s_2 and s_3 as shown in Fig. 1. The target s_1 is moving in the direction of A-B, s_2 in the direction of C-D and s_3 in the direction of E-F. The pursuer starting from the origin O, should visit each target only once and return to the origin O in the minimum possible time.

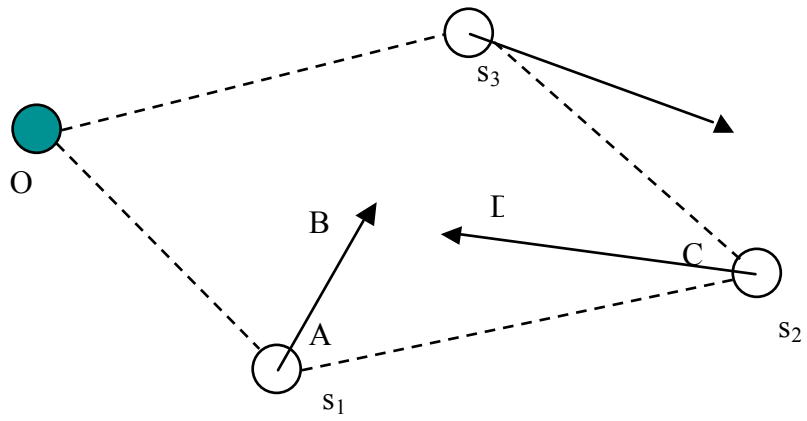

Figure 1. An example of NTSP with three cities.

Helvig et al (2003) pointed out that minimizing the travel time is equivalent to minimizing the distance travelled for the stationary TSP (STSP). However, in the case of NTSP, these two objectives are very different, each leading to distinct properties, strategies, and results.

3. Solution approaches

Stationary TSP (STSP) is a well-known NP hard problem. It is unlikely that an optimization algorithm would find optimal tours for a STSP with more than 30 cities in the general case in a reasonable time. Most algorithms developed for STSP are either approximation algorithms or heuristics. Christofiedes' algorithm is one of the best approximation methods for STSP that can guarantee to find the solutions with the error rate 3/2 (Johnson and Papadimitriou, 1985). However, if the number of cities is large, the error rate is too high. Johnson and Papadimitriou (1985) studied the possibility of polynomial-time heuristics that provide good guarantees for all (symmetric) STSP instances and showed that this can only happen in the unlikely event that P=NP. Heuristic techniques such as Simulated Annealing (SA) (Budinich, 1996; Kirkpatrick et al, 1983; and Jeong and Kim, 1991), Genetic Algorithms (GAs) (Choi et al, 2003 and Moon et al, 2002), Tabu search (TS) (Carlton and Barnes, 1996 and Knox, 1994) and Ant Colony Optimization (ACO) (Stutzle and Hoos, 2000 and Branke and Guntsch, 2003) are applied to solve many instances of STSP. In this paper, we are interested in solution approaches of NTSP.

NTSP expands the domain and complexity of the classical TSP. The stationary targets/cities become non-stationary and minimizing the traveling distance (or time) becomes harder. Similar to the original TSP, the NTSP is NP-complete, where it is unlikely that polynomial-time exact algorithms can be developed.

Formin and Lingas (2002) provides a $(2+\varepsilon)$-approximation algorithm for the NTSP. Helvig et al (2003) presented exact algorithms for three special cases of NTSP: all targets have the same direction, only limited number of moving targets and all targets have the same speed. However their algorithms are not suitable for the general cases of NTSP. From Hammar and Nilsson (1999), it is clear that the approximation methods are only suitable for some special cases of NTSP rather than the general NTSP cases. In this paper, we develop a GA based algorithm for NTSP.

4. The algorithm

We used a straightforward GA algorithm. The algorithm starts with initializing the population with randomly generated (but feasible) solutions. Each solution is then evaluated by simulating the movements of each target and calculating the anticipated interception points. The reproduction loop then takes place through selecting and crossing-over parents, and mutating the resultant children. If illegal solutions are generated, a repair operator is applied by randomly fixing the duplicated parts in a solution. Solutions are then evaluated and the reproduction cycle continues. The proposed GA based algorithm for NTSP is as follows:

```
Begin
    t _ 0
    initialize P (t)
    predict interception and evaluate P (t)
    while ( not terminate-condition ) do
        begin
```



```
                t  _ t + 1
                select P (t+1) from P (t)
                alter P (t+1)
                apply repair operator to P (t+1)
                predict interception and evaluate P (t+1)
            end
        end
```

For convenience of representation, we choose the integer coded GA rather than the traditional binary coding. First, we create a number of tours each with an order of cities to be visited for example {1, 3, 4 … *n*}. Then we calculate the fitness of each tour (total traveling time). The fitness calculation in NTSP is not as simple as STSP. To explain the fitness calculation, let us consider the example of Figure 1 where we define a tour as O-s_1-s_2-s_3-O. Calculate the time required for each pair of cities (targets) in the tour as follows:

- Calculate the minimum time required from O to s_1. Since s_1 is moving at a constant velocity v_1, the pursuer must meet s_1 at location A_1 instead of A (see Fig. 2). That means the pursuer must travel towards A_1, to minimize the travel time, which is not difficult to find as the direction and velocity of the target s_1 and the speed of pursuer are known and constant. In our calculation, we use the maximum speed of the pursuer. The assumption here is that the pursuer speed is higher than the speed of any target. If this assumption is violated, the definition of NTSP will need to change because in some cases, the pursuer may not be able to intercept with the target. Suppose the travel time from O to s_1 is T_1. Once the pursuer reaches A_1 (after time T_1), the targets s_2 and s_3 will reach location C_1 and E_1 respectively.

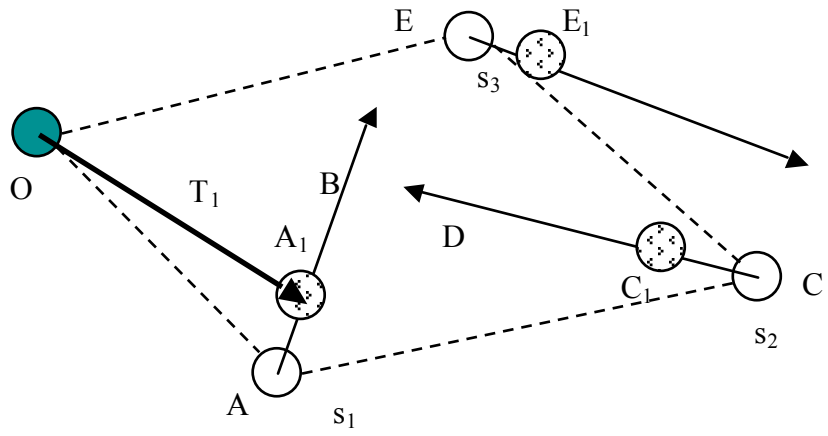

Figure 2. Movement from O to s_1 (at time T_1).

- The pursuer will take time T_2 to travel from A_1 to the new location of s_2 which is C_2 as of Fig. 3. The target s_3 will reach the location E_2 by that time.

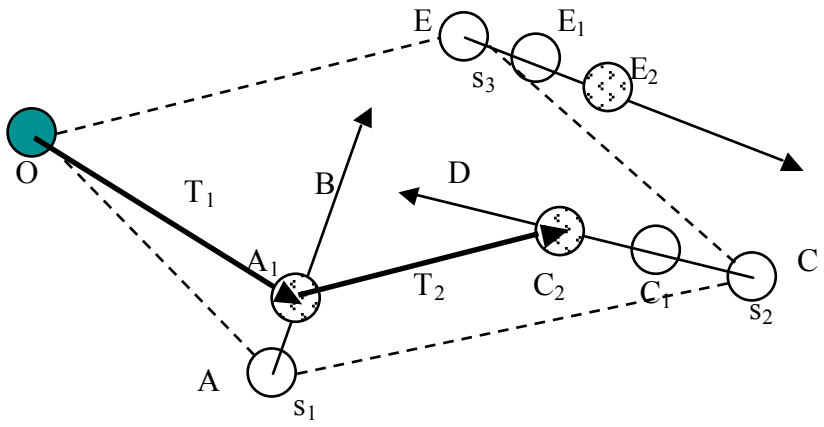

Figure 3. Movement from A_1 to s_2 (at time $T_1 + T_2$).

- The pursuer will take time T_3 to travel from C_2 to the new location of s_3 which is E_3 as of Fig. 4.
- The pursuer will take time T_4 to travel from E_3 to the origin *O*.
- The fitness of the tour is $f = T_1+T_2+T_3+T_4$.

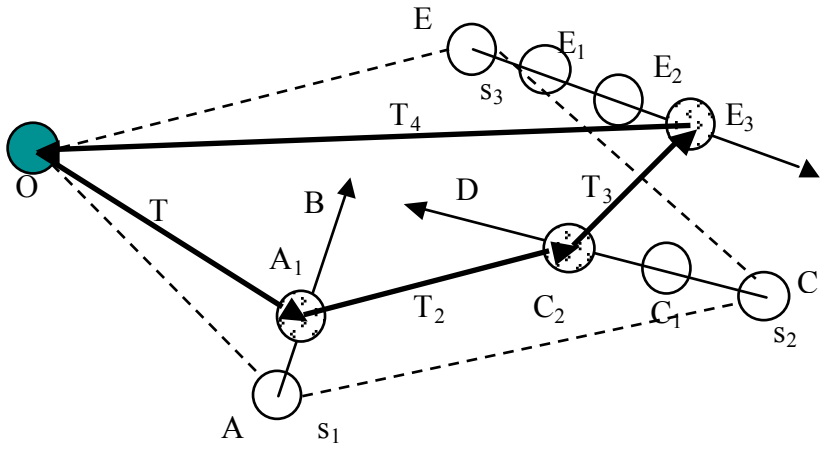

Figure 4. Movement from C_2 to s_3 and s_3 to O.

In our algorithm, we use two different crossovers: Order Crossover (OX) and Cycle Crossover (CX). Both crossover operators guarantee that if the two parents are valid tours, the child is also a valid tour. In OX, two cut points are randomly chosen on the parent chromosomes. In order to create an offspring, the string between the two cut points in the first parent is first copied to the offspring, then the remaining position are filled by considering the sequence of activities in the second parent, staring after the second cut point (when the end of the chromosome is reached, the sequence continues at position 1).

CX focuses on subsets of cities that occupy the same subset of positions in both parents. Then, these cities are copied from the first parent to the offspring (at the same position), and

the remaining positions are filled with the cities of the second parent. In this way, the position of each city is inherited from one of the two parents. However, many edges can be broken in the process, because the initial subset of cities is not necessarily located at consecutive positions in the parent tours.

We use the swap_2 as mutation operator. Two cities are randomly selected and swapped (i.e., their positions are exchanged). This mutation operator is the closest in philosophy to the original mutation operator, because it only slightly modifies the original tour with 4 edges at the most.

Finally we apply a repair mechanism to repair any infeasible tours created in any stage of the algorithm such as the pursuer visits any cities twice and/or miss any other cities. The repair mechanism is used mainly for the initial population since all operators guarantee to generate valid tours.

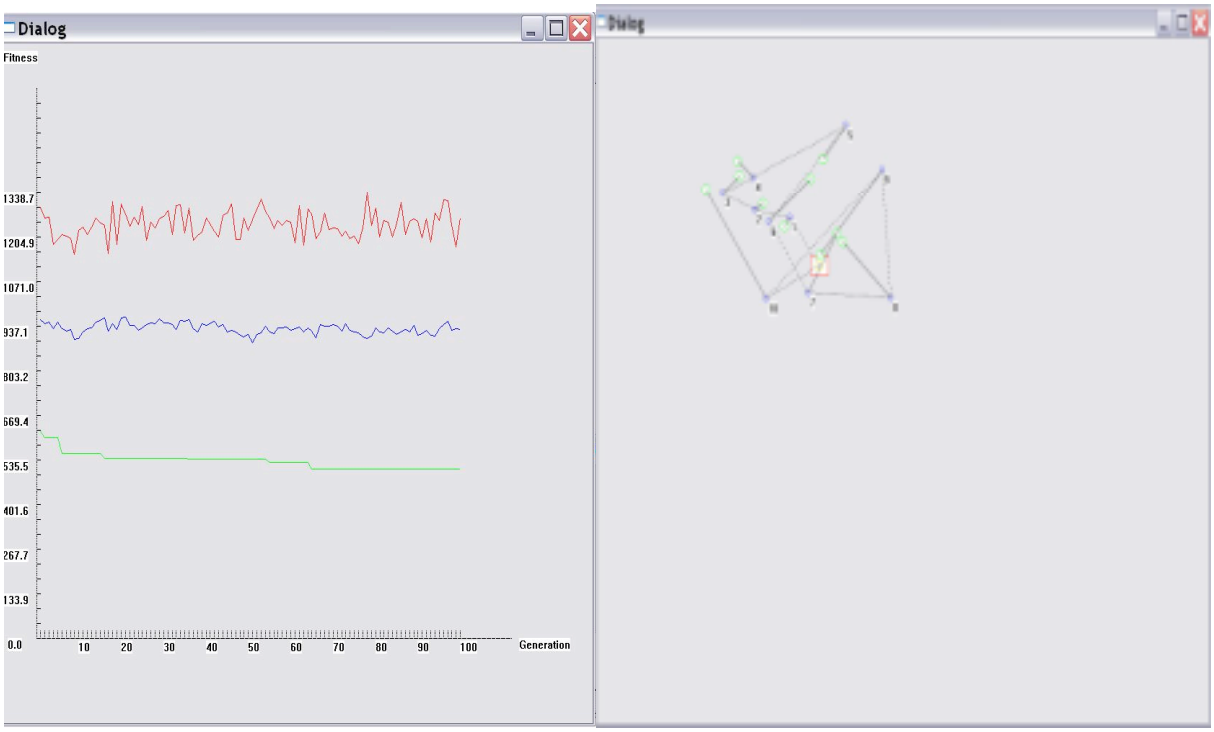

Figure 5. Two screen dumps of the tracking moving target problem. The left figure shows the progress of the evolutionary computation model (worst fitness (top line), average fitness (middle line), and best fitness (bottom line)) and the right figure shows the sequence of interceptions.

5. Experimental setup

We have generated 30 NTSP instances of 10 cities randomly. The pursuer's location and speed, and the location, speed and direction of movement for each city were generated within the following ranges.

Table 1. Test problems' data.

	Initial location		Speed	Direction of movement	
	x-axis	y-axis		x-axis	y-axis
Pursuer	0-100	0-100	20-40	-	-
City	0-100	0-100	10-20	-1 to +1	-1 to +1

As an example, one problem instance is shown in Tables 2 and 3.

Table 2. Details of pursuer.

Number of Cities:	10
Initial position of pursuer (x-axis)	98.5424
Initial position of pursuer (y-axis)	51.4723
The speed of pursuer:	30

Table 3. Details of targets.

	Initial position		Moving direction		Speed
City ID	X-axis	Y-axis	X-axis	Y-axis	
1	18.8665	15.6839	-0.886946	-0.461873	17
2	2.81664	8.91639	0.270585	-0.962696	13
3	18.2772	20.6597	0.676864	0.736108	14
4	83.9867	29.4462	-0.955202	0.295954	11
5	46.7837	61.3042	-0.457243	-0.889342	10
6	84.3494	39.8209	0.765682	0.643219	15
7	59.226	94.6413	-0.756471	0.654027	16
8	73.9737	61.7654	-0.0791881	-0.99686	15
9	13.6156	76.45	0.333373	-0.942795	13
10	10.8492	51.9188	-0.824517	-0.565837	13

We use the following parameters in our experiments:

Population size	30
Probability of crossover	0.8
Probability of mutation	0.1
Number of generations in each run	30

Number of independent runs	30

6. Results and discussions

For each problem instance, we have recoded the best fitness, worst fitness, and mean fitness with standard deviation for both OX and CX. The experimental results of 30 NTSP instances for both CX and OX are analysed and reported in the following table.

Table 4. Comparing Crossover operators.

Crossover operator	Number of instances with better fitness	Percentage of total
CX	5	16.67
OX	7	23.33

By the best fitness value, we mean the minimum of 30 minimum values obtained from each of 30 independent runs (one minimum for each run) for a given NTSP instance. A total of 18 instances out of 30 produced exactly the same best fitness value using either crossover operator. In the remaining 12 instances, CX operator produces better results for 5 instances and OX operator produces better results for 7 instances. Although from the limited experimental results it is hard to say which operator is better (CX or OX), we present the minimum and the average of minimum fitness values for 12 competing instances in Fig. 5.

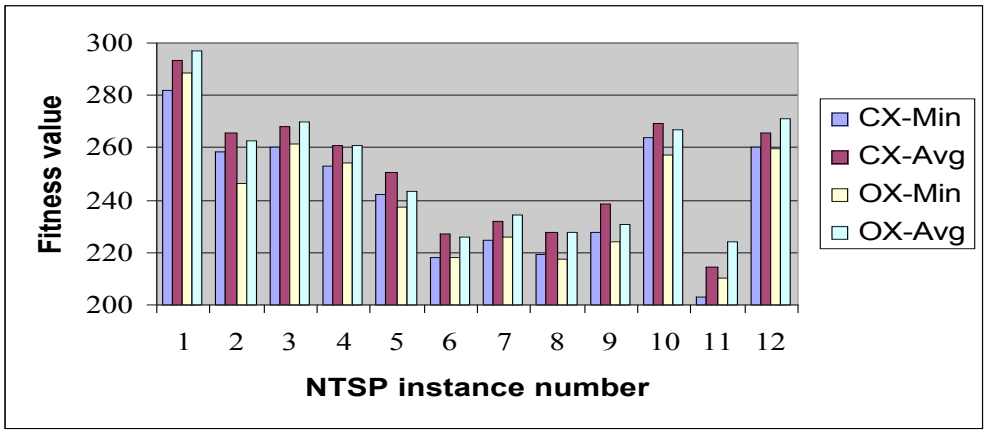

Figure 6. The best and average-best fitness for CX and OX.

By the average-best fitness value, we mean the average of 30 minimum fitness values obtained from 30 independent runs (minimum of each run) for a given NTSP instance. As we can see in the Figure 6, if the fitness is lower for a crossover operator, the corresponding average-best fitness is consistently lower. That ensures again there is no clear winner so far.

7. Conclusion

In this paper, we have introduced a non-stationary version of the classical traveling salesman problem. We have developed a GA based methodology for solving NTSP problems. To demonstrate the use of our developed algorithm, we have solved 30 randomly generated NTSP instances using two different crossover operators.

References

Branke, J. and M. Guntsch (2003), New ideas for applying ant colony optimization to the probabilistic TSP, *Applications of Evolutionary Computing*, 2611: 165-175.

Budinich, M. (1996), A self-organizing neural network for the traveling salesman problem that is competitive with simulated annealing, *Neural Computation*, 8(2): 416-424.

Carlton, W.B. and J.W. Barnes (1996), Solving the traveling-salesman problem with time windows using tabu search, *IIE Transactions*, 28(8): 617-629.

Choi, I.C., S.I. Kim, and H.S. Kim (2003), A genetic algorithm with a mixed region search for the asymmetric traveling salesman problem, *Computers & Operations Research*, 30(5): 773-786.

Fomin, F.V. and A. Lingas (2002), Approximation algorithms for time-dependent orienteering, *Information Processing Letters*, 83(2): 57-62.

Hammar, M. and B. Nilsson (1999), Approximation Results for Kinetic Variants of TSP, *In Automata, Languages and Programming: 26th International Colloquium(ICALP'99)*, Prague, Czech: Springer.

Helvig, C.S., G. Robins, and A. Zelikovsky (2003), The moving-target traveling salesman problem, *Journal of Algorithms*, 49(1): 153-174.

Huang, L., C.G. Zhou, and K.P. Wang (2003), Hybrid ant colony algorithm for traveling salesman problem, *Progress in Natural Science*, 13(4): 295-299.

Jeong, C.S. and M.H. Kim (1991), Fast Parallel Simulated Annealing for Traveling Salesman Problem on Simd-Machines with Linear Interconnections, *Parallel Computing*, 17(2-3): 221-228.

Johnson, D.S. and C.H. Papadimitriou (1985), Performance guarantees for heuristic, In *The Traveling salesman problem: a guided tour of combinatorial optimization*, L. Eugene L, Editor. Wiley: Chichester (West Sussex]); New York, 145-180.

Knox, J. (1994), Tabu Search Performance on the Symmetrical Traveling Salesman Problem, *Computers & Operations Research*, 21(8): 867-876.

S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Science 220 (1983) 671-680.

Moon, C., et al. (2002), An efficient genetic algorithm for the traveling salesman problem with precedence constraints, *European Journal of Operational Research*, 140(3): 606-617.

Stutzle, T. and H.H. Hoos (2000), MAX-MIN Ant System, *Future Generation Computer Systems*, 16(8): 889-914.

Zhou, A., L. Kang, and Z. Yan (2003), Solving Dynamic TSP with Evolutionary Approach in Real Time, *Proceedings of IEEE-CEC2003*, 2: 951-957.

