A New Approach to Solving Dynamic Traveling Salesman Problems

Changhe Li[1] Ming Yang[1] Lishan Kang[1]

[1]China University of Geosciences(Wuhan) School of Computer
430074 Wuhan,P.R.China
lchwfx@yahoo.com,yangming0702@gmail.com,kang_whu@yahoo.com

Abstract. The Traveling Salesman Problem (TSP) is one of the classic NP hard optimization problems. The Dynamic TSP (DTSP) is arguably even more difficult than general static TSP. Moreover the DTSP is widely applicable to real-world applications, arguably even more than its static equivalent. However its investigation is only in the preliminary stages. There are many open questions to be investigated. This paper proposes an effective algorithm to solve DTSP. Experiments showed that this algorithm is effective, as it can find very high quality solutions using only a very short time step.

1 Introduction

With the development of computer science and communication technology, from highly centralized computing through distributed computing to today's highly mobile computing, computing environments have changed a great deal. Key research challenges they face in common are the optimization of dynamic networks, arising from network planning and designing, load-balance routing and traffic management. However, guaranteeing that these systems run effectively and reliably is a difficult problem. It leads to a very important theoretical mathematical model: the Dynamic TSP (DTSP).

Because of the characteristics of DTSP itself, the solutions to general static TSPs are usually unsuitable for DTSP. Most cost too much time to gain good solutions, so the general algorithms are inefficient. Though a number of authors have researched [1][2][3][4] the DTSP, since it was proposed by Psaraftis[5], exploration of the DTSP is still in the preliminary stages, and many open questions need to be discussed. The ultimate (but unobtainable) goal is to find an optimum solution at every moment, as time progresses. In fact, if you want to get better solutions, the efficiency will be lower, and conversely, if you require quick solutions, their quality will reduce that is to say the two goals (solution quality and time response)are in conflict. Since we can't get an optimum solution at every instant, we can solve the problem in discrete time steps, finding good solutions in a time step as short as possible.

In this paper, we will introduce an improved Inver-Over[6] algorithm(GSInver-Over) based on a gene pool. Generally, we find that using a gene pool, which reduces the search space sharply, gives solutions much more rapidly, without degradation of the solution quality. Thus it dramatically improves the system performance in the combined objectives. The GSInver-Over algorithms can improve the individuals using information either from other individuals or from gene pools. We augment this with elastic relaxation as a local search method, which permits the rapid evolution of variants of individuals which were successful in previous situations. Our experiments show that these operators can provide highly satisfactory results.

In the remainder of this paper, there are five sections: (1) description of DTSP; (2) design of the gene pool; (3) the detailed algorithms; (4) analysis of the results; (5) summary and conclusions.

2 Description of DTSP

Definition 1

A dynamic TSP(DTSP) is a TSP determined by a dynamic cost (distance) matrix as follows:

$$D(t) = \{d_{ij}(t)\}_{n(t)\times n(t)} \qquad \textbf{(1)}$$

where $d_{ij}(t)$ is the cost from city(node) c_i to city c_j, and t is the real-world time. In this definition, the number of cities $n(t)$ and the cost matrix are time-dependent. The Dynamic Traveling Salesman Problem is to find a minimum-cost route containing the all the $n(t)$ nodes.

Definition 2 DTSP:

Given all $n(t)$ $(P_1, P_2, ..., P_{n(t)})$ points, and the corresponding cost matrix $D = \{d_{ij}(t)\}, i, j = 1, 2, ..., n(t)$, find a minimum-cost route containing all the $n(t)$ points, where t stands for the moment of time t; $d_{ij}(t)$ stands for the distance between the objective point P_i and the objective point P_j, and $d_{ij}(t) = d_{ji}(t)$.

For example：

$$Min(d(T(t))) = \min(\sum_{i=1}^{n(t)} d_{T_i, T_{i+1}}(t)) \qquad \textbf{(2)}$$

where $T \in \{1, 2, ..., n(t)\}$ if $i \neq j$, then $T_i \neq T_j$, $T_{n(t)+1} = T_1$

In definition 1, we deem the change of a DTSP's cost matrix with time as a continuous process. Practically, we discretize this change process. Thus, A DTSP becomes a series of optimization problems:

$$D(t_k) = \{d_{ij}(t_k)\}_{n(t_k)\times n(t_k)} \qquad \textbf{(3)}$$

$k = 0, 1, 2, ..., m-1$, with time windows [t_k, t_{k+1}], where $\{t_k\}_{i=0}^{m}$ is a sequence of real world time sampling points.

3 Design of Gene Pool

Heuristic rules can dramatically affect the efficiency of DTSP algorithms. The TSP is an NP-hard problem, and adding only one node, will increase the candidate search space by $n \times n!$. Thus it is impossible to search all the candidate individuals. One useful heuristic derives from the fact that most of the edges in a minimum-cost route will join nearby vertices. So it is generally desirable for the elements in the gene pool to select from edges which go to nearby vertices. Unfortunately, this heuristic is often violated: for hard TSPs, a small proportion of the edges in optimal routes will need to connect distant vertices. If the heuristic is too rigidly applied, some of the edges in the optimal route won't exist in the gene pool. So a heuristic method based on minimising local distances seems reasonable, but in fact, this kind of restriction usually results in bad performances. We describe an alternative heuristic for the design of the gene pool. It is based on the concept of α-nearness [7], which derives from Minimum Spanning Trees (MSTs). The α-length α(i,j) of an edge $<v_i,v_j>$ can be defined as the difference in length between the true MST, and the length of the 1-tree which is constrained to contain $<v_i, v_j>$.

$$\alpha(i,j) = L(T^+(i,j)) - L(T) \tag{4}$$

where T is an any given MST of length L(T),$T^+(i,j)$ is a 1-tree that contain the edge $< v_i, v_j >$,that is, given a MST of length L(T),α(i,j) is the increase length of a 1-tree required to contain the edge $< v_i, v_j >$.

It is easy to see that: α(i,j)≥0 and α(i,j)＝0 when the edge $< v_i, v_j >$ belong to T. It can compute α(i,j) in the following rules:

(1) if the edge $< v_i, v_j >$ belong to T, then α(i,j)=0.

(2) Otherwise, insert the edge $< v_i, v_j >$ into T, this will create a circle containing the edge $< v_i, v_j >$,then α(i,j) is the length difference between the longest edge of the circle and the edge $< v_i, v_j >$.

The gene pool is a candidate set of some most promising edges. The candidate set may, for example, contain k α-nearness edges incident to each node. Generally speaking, the experience value of k is 6 to 9.we set k=8 in CHN144 problem[8].

Through experiments, it can be shown experimentally that 50%-80% (i.e the experiment result of table 1 of instances in TSPLIB) of the connections in an optimal TSP solution are also in the minimum spanning tree. This is a far larger proportion than the proportion of the n shortest edges. Thus we expect that a gene pool constructed based on α-nearness may perform better than a gene pool constructed on the distance. It should be possible to use a smaller gene pool while maintaining the solution quality. Taking the CHN144 problem for example, 76.3% of the connections in the best known solution are also edges in the minimum spanning tree. If we bias the gene pool based on the α-nearness, we expect that it will better match the optimal solution. That is to say, we expect that a gene pool that probabilistically includes elements close to members of the MST will also include elements close to members of the optimum TSP circuit. The gene pool based on the α-nearness has another remarkable advantage that it is independent of instance scale, it means ,when the instance scale increases ,the size of gene pool won't increase. This character of gene pool is much suitable to DTSP.

Table 1. Instances in TSPLIB

CHN144	a280	pr439	u574	u724	rl1323
76.3%	75.7%	79.1%	75.6%	75.2%	89.2%

4 Introducing the Algorithms

Operator design is the key to solving TSPs. A vast range of operators have already been proposed (for example: λ-Opt[9], LK[10], Inver-Over), and we anticipate that this trend will continue. Based on performance, many of these local-search inspired operators are superior to the traditional mutation, crossover and inversion operators. In this paper, we adopt a form of improved Inver-Over operator. We propose a highly efficient dynamic evolution algorithm based on elastic.

4.1 GSInver-Over Operator

Inver-Over is a high-efficiency search operator based on inversion, and having a recombination aspect. It can fully utilize information from other individuals in the population to constantly renew itself. This gives it better search ability than many other operators (within a certain range of problems), yet the complexity of algorithm is low. We can say Inver-Over is a highly adaptable operator which has very effective selection pressure.

However Inver-Over operator has its own constrains, experiments show that reducing the inversion times can sharply increase the convergence, this is favourable to DTSP. We set the maximal inversion times Max_time in the GSInver-Over operator, when the inversion times surpass Max_time, end the algorithm. The search environment has increased and it can get useful heuristic information not only from other individuals, but also from the gene pool. The choice of matching individuals is not random, but biased toward better individuals in the population. This reduces the probability of incorporating bad information that damages the individual. The inversion operator is more rationally designed, incorporating knowledge about the directionality of the route, further improving the performance of the algorithm. Based on the above improvements, the GSInver-Over performed substantially better than the original algorithm. the algorithm for our GSInver-Over operator is as follows:
GSInver-Over Operator:

1. Select a gene g from individual S randomly and set S′=S;
2. If the number p generated randomly is less than p1, then select the gene g′ from the gene pool of g;
3. Else if p<p2 select an individual randomly from some best individuals and g′ is the gene that is next to g in the selected individual;
4. Else select next gene g′ randomly from other genes;

```
5. If the next gene or the previous gene of g in S′ is g′, then go to step 9;
6. Inverse the section from the next gene of g to g′ in S′;
7. counter++, if counter > Max_time, then go to step 9;
8. g= g′ and go to step 2;
9. If the fitness of S′ is better than S, then replace S with S′;
```

4.2 The Dynamic Elastic Operator

The changes of a node include three cases: the node disappears, the node appears and the position of the node changes. If the node disappears, it will be deleted directly, then link the two adjacent nodes of the deleted node. if the node appear, find the nearest node to the appearing node, then insert it to the tour that minimize the total length. if the node position changes, it can be seen as the combination of the two former cases. The dynamic-elastic operator is very simple in concept, but we find it is an effective local search operator.

```
Dynamic Elastic Operator
 1. Delete the node c and link the cities adjacent to c;
 2. Find the nearest node c* to c in the current individual;
 3. Insert c next to C*, on the side that minimize the total length;
```

4.3 Main Program Loop

In the main program loop, we use a difference list Dlist to store the information of changed nodes. Note that ΔT is the time step.

```
1. Initialize the population;
2. If Dlist is not empty goto 3, else goto 5;
3. Update gene pool;
4. Dynamic-elactic ();
5. For each individual in the population,do
   GSInver-Over ();
   Optimizing();
6. If the ΔT>0 goto 5;
7. If not termination condition goto 2;
```

When some nodes change at time t, it needs to update the gene pool, that is to say, create a new MST of the new nodes topology of time t, then construct a gene pool according to α-nearness.

5 Experiments with CHN146+3

We tested our algorithm in a relatively difficult dynamic environment, adapted from the well-known CHN145[11] static TSP benchmark. The problem has 146 static locations (145 Chinese cities, plus a geo-stationary satellite) and three mobile locations, two satellites in polar orbit and one in equatorial orbit (fig. 1).

In dynamic optimisation experiments, since the results represent a trade-off between solution quality, computational cost and problem dynamics, it is important to specify the computational environment in which experiments were conducted. Our experimental environment consisted of the following: CPU: Intel C4 1.7GHz, RAM:256MB. We measure the offline error ē and μ as our quality metric:

$$\bar{e}(t_k) = d(\pi(t_k)) - d(\bar{\pi}(t_k)) \qquad \textbf{(5)}$$

$$\mu(t_k) = \bar{e}(t_k) / d(\bar{\pi}(t_k)) \qquad \textbf{(6)}$$

where $d(\bar{\pi}(t_k))$ is the best tour obtained by a TSP solver (which is assumed to be good enough to find an optimal solution for the static TSP) $d(\bar{\pi}(t_k))$ is the best tour obtained by our DTSP solver. Together with:
Maximum error:

$$e_m = \max_{k=0,\cdots,m} \{\bar{e}(t_k)\} \qquad \mu_m = \max_{k=0,\cdots,m} \{\mu(t_k)\} \qquad \textbf{(7)}$$

Minimum error:

$$e_r = \min_{k=0,\cdots,m} \{\bar{e}(t_k)\} \qquad \mu_r = \min_{k=0,\cdots,m} \{\mu(t_k)\} \qquad \textbf{(8)}$$

Average error:

$$e_a = \frac{1}{m+1}\sum_{t=0}^{m}(\bar{e}(t_k)) \qquad \mu_a = \frac{1}{m+1}\sum_{t=0}^{m}(\mu(t_k)) \qquad \textbf{(9)}$$

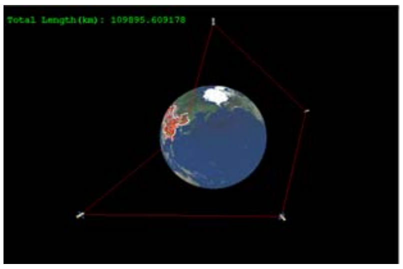

Fig. 1. Experiments with CHN146+3

120 sample time-points in the period of the satellites were performed for the experiments. The results are given in table 2 with ΔT ranging from 0.059s to 1.3s. Fig. 2 to fig. 5 are error curves respectively for ΔT=0.059s, ΔT=0.326s, ΔT=0.579s and ΔT=0.982s.

From the experiments, we can see that, when ΔT is very small, the result is relatively poor. As ΔT increases, the maximal and average errors decrease, and the solution quality improves showing the stability of the algorithm, and its rapid convergence. The experiments also demonstrate the conflict between the two DTSP goals, requiring a compromise through the choice of ΔT. With the exception of the shortest time interval, the data in table 2 are generally acceptable, with the average errors being under 1%, and the maximal errors less than 2%.

Table 2. Error of experiments

ΔT(s)	e_m(km)	μ_m(%)	e_r (km)	μ_r(%)	e_a(km)	μ_a(%)
0.059	1742	1.487	206	0.199	796	0.727
0.222	2020	1.722	773	0.062	406	0.372
0.326	1310	1.122	0	0	289	0.264
0.481	1014	0.866	0	0	283	0.257
0.579	1038	0.884	0	0	203	0.187
0.982	899	0.770	0	0	240	0.218
1.300	1514	1.297	0	0	188	0.170

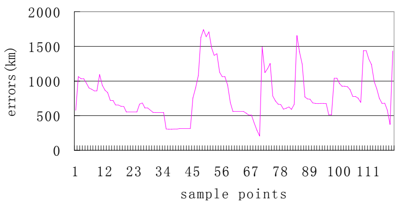

Fig. 2. Error Curve for ΔT = 0.059s

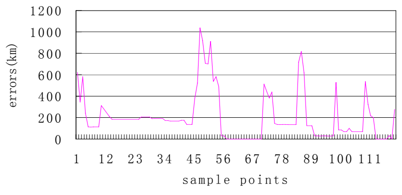

Fig. 4. Error Curve for ΔT = 0.579s

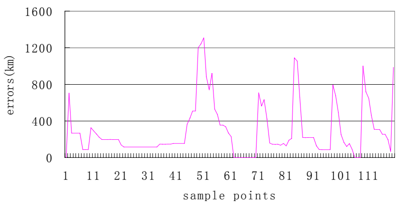

Fig. 3. Error Curve for ΔT = 0.326s

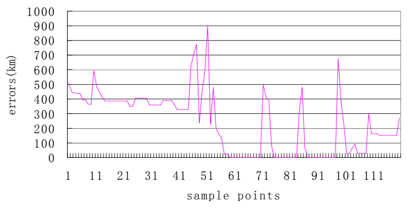

Fig.5. Error Curve for ΔT = 0.982s

6 Conclusions

In this paper, we analyze the DTSP which can be seen as a two-objective problem by trading off the quality of the result and the reaction time. We propose a solution based on a gene pool, which greatly reduces the search space without degradation of the solution quality. By adding the improved GSInver-Over Operator, we were able to significantly improve the efficiency of the algorithm. Adding the elastic relaxation method as a local search operator improves the system's real-time reaction ability.

7 Acknowledgements

This work was supported by National Natural Science Foundation of China (NO. 60473081). We would like to thank Bob McKay, of Seoul National University, Korea, for contributions to the presentation of this work.

8 References

1. C.J.Eyckelhof and M.Snoek. Ant Systems for a Dynamic TSP -Ants caught in a traffic jam. In3rd International Workshop on Ant Algorithms(2002)
2. Z.C.Huang,X.L.Hu and S.D.Chen. Dynamic Traveling Salesman Problem based on Evolutionary Computation. In Congress on Evolutionary Computation(CEC'01),IEEE Press (2001)1283–1288
3. Allan Larsen. The Dynamic Vehicle Routing Problem. Ph.D theis,Department of MathematicalModelling (IMM) at the Technical University of Denmark(DTU)(2001)
4. A.M.Zhou, L.S.Kang and Z.Y.Yan. Solving Dynamic TSP with Evolutionary Approach in Real Time. Proceedings of the Congress on Evolutionary Computation, Canberra, Austrilia, 8-12, December 2003, IEEE Press(2003) 951–957
5. H.N.Psaraftis. Dynamic vehicle routing problems. In Vehicles Routing: Methods and Studies,B.L.Golden and A.A.Assad(eds),Elsevier Science Publishers(1988)
6. T.Guo and Z.Michalewize. Inver-Over operator for the TSP. In Parallel Problem Sovling from Nature(1998)
7. Keld Helsgaun,An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic，Department of Computer Science Roskilde University.
8. L.S.Kang, Y.Xie,S.Y.You,etc. Nonnumerical Parallel Algorithms:Simulated Annealing Algorithm. Being:Science Press(1997)
9. S. Lin,"Computer Solutions of the Traveling Salesman Problem",Bell System Tech. J., 44, (1965) 2245–2269
10. S. Lin & B. W. Kernighan,An Effective Heuristic Algorithm for the Traveling-Salesman Problem(1973)
11. Lishan Kang, Aimin Zhou, Bob McKay,Yan Li Zhuo Kang ,Benchmarking Algorithms for Dynamic Travelling Salesman Problems, Benchmarking Algorithms for Dynamic Travelling Salesman Problems, Proceedings, Congress on Evolutionary Computation, Portland, Oregon(2004)