An Improved Dynamic Programming Algorithm for Bitonic TSP

LI Jian
Foundation Department
The PLA Foreign Languages University
Luoyang, China
e-mail:maomaotfntfn@163.com

***Abstract*—This paper puts forward an improved dynamic programming algorithm for bitonic TSP and it proves to be correct. Divide the whole loop into right-and-left parts through analyzing the key point connecting to the last one directly; then construct a new optimal sub-structure and recursion. The time complexity of the new algorithm is $O(n^2)$ and the space complexity is $O(n)$; while both the time and space complexities of the classical algorithm are $O(n^2)$. Experiment results showed that the new algorithm not only reduces the space requirement greatly but also increases the computing speed by 2-3 times compared with the classical algorithm.**

Keywords-dynamic programming; bitonic TSP; optimal sub-structure; space complexity

I. INTRODUCTION

TSP question can be described as this [1]: finding out the shortest loop to connect the n points on a plane on the premise that each point appears only once. From the essence, the question is mainly equal to finding out the optimal permutation among all the possible permutations of the n points, completely NP [2]. *J.L. Bentley* [3] suggests that the question can be simplified when we only consider the bitonic loop, which can be described as this: starting from the left-most point, strictly passing some of the points from the left to the right and finally reaching the right-most point, after which coming back passing the rest points. For bitonic TSP, it is proved that finding out an algorithm within polynomial time is feasible [4].

Dynamic programming is a powerful algorithm design method and widely used in combinatorial optimization problem [5, 6]. This paper will firstly introduce both the classic and improved algorithms for bitonic TSP and then make a comparison between them.

II. THE CLASSIC ALGORITHM

The thinking of classic algorithm to the bitonic TSP question is [6]: a man starts from the left-most point and walks strictly from the left to the right; then comes back from the right to the left. The process above equals that two men starts from the left-most point and strictly passes different paths to the right-most point.

To simplify the illustration, we suppose the two men are A and B, while A is always ahead of B (B is closer to the right-most point). Let's set $M[i, j]$ indicating the length of the shortest bitonic path when A walks to point i and B walks to j. From the supposition, it is known that $j \le i$, and we should notice that $M[i,j]$ means all the first i points having been passed without anyone skipped. And we suppose that $D[i,j]$ indicates the straight-line distance between i and j. Now we will have these recursions:

When $i = j$, $M[i, j] = M[i, i-1] + D[i, j]$

When $i > j+1$, $M[i, j] = M[i-1, j] + D[i, i-1]$

When $i = j+1$, $M[i, j] = \min_{1 \le k \le j} \{M[j, k] + D[i, k]\}$

While implementing the algorithm above, a two-dimension array can be used as the vessel to store the results. And a dual loop can help accomplish the recursive process. And it is easy to see that both the space complexity and the time complexity are $O(n^2)$.

III. PRINCIPLE OF THE IMPROVED ALGORITHM

A. Relevant Definitions

The algorithm mentioned above has already been widespread as the classic algorithm to the bitonic TSP questions. By constructing a new optimal substructure, the dynamic programming algorithm can be improved by reducing its space complexity. At the beginning, this discourse will bring in some definitions for a better understanding of the following illustration.

Definition 1: If a bitonic TSP loop includes n points, number them in the order of left-to-right. For example, name number i point P_i. And if P_i and P_j are directly linked within the loop, name the edge between P_i and P_j as *Edge*(i, j).

Definition 2: Within a bitonic TSP loop, there are two links connecting points from number 1 to n. One is called up-link while the other is down-link. However, we should notice that up-link does not always stay the upside of the down-link, vice versa. In non-optimal loop, they two could also intersect.

Definition 3: Within a loop including n points, there are two points connecting to the right-most point. One always is the P_{n-1} while the other can be any one from P_1 to P_{n-2}. The point that directly connects the right-most point (except P_{n-1}) is the key point of the loop.

B. Local Optimal Solution

In fact, using the dynamic programming algorithm to find the solution is a recursive process because when solving the number n question, we usually assume that the question number n-1 has been worked out[7]. Let's suppose the question's scale increases progressively in the order of

Published by Atlantis Press, Paris, France.

left-to-right, in which new points could only be added at the right-most point. $M[i]$ indicates the length of the shortest bitonic TSP loop constructed by the first n points. And $D[i,j]$ indicates the distance between P_i and P_j. For a bitonic TSP loop of n points, its optimal loop is fixed once the key point determined.

Lemma 1: For the n points on a plane (See Fig. 1), when P_k is the key point (n>2,1<k≤n-2), the length of the optimal bitonic TSP loop is :

$$D[k,n]+\sum_{i=k+1}^{n-1} D[i,i+1]+M[k+1]-D[k,k+1]$$

Proof: we can draw an auxiliary line to link point P_k and P_{k+1}(see the dotted line in Fig. 1) and the whole loop can be divided into two parts, namely left-half loop and right-half loop. After that we can analyze the structure of the optimal loop:

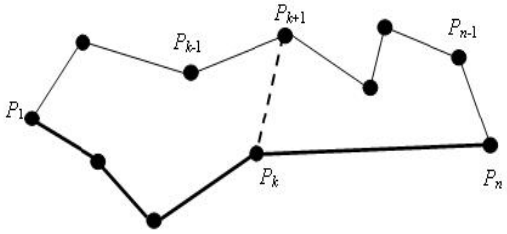

Figure 1. Sketch map of the improved algorithm

P_k as the key point directly connects P_n. *Edge*(k,n) must be within the loop and its length is $D[k,n]$. Assume that the edge consisting *Edge*(k,n) is the down-link(the proof of the up-link is the same), so that all the points from P_k to P_{k+1} are in the up-link. Assume P_l (k<l<n) exists in the down-link; then according to the bitonic features, P_n links P_1 before P_k because P_1 is at the right hand of P_k. However, it is paradoxical with the premise (P_k connects P_n directly); so the assumption is false. By connecting points orderly from P_{k+1} to P_n, the up-link of the right-hand loop is fixed and its length is :

$$\sum_{i=k+1}^{n-1} D[i,i+1]$$

When the key point is determined, the right-half loop is fixed and it will be the shortest. So we only need to let the left-half loop fulfill the bitonic features and let its path be the shortest. The left-half loop is the shortest if it meets the requirement that: on the base of the first k+1 points, constructing the optimal loop and removing *Edge* (k,k+1) whose length is $M[k+1]$-$D[k,k+1]$. Assume that there is another path that is shorter than the one mentioned above; then on the base of it, let's add *Edge* (k,k+1). After that we can get a bitonic TSP loop of the first k+1 points, which is shorter than the known optimal loop. As a result, the assumption is also paradoxical and false.

The whole length of the current optimal loop is the sum of each part's. So far, the proof of lemma 1is finished.

C. Global Optimal Solution

In lemma 1, assume that the key point of bitonic TSP loop is determined and the solution is locally optimal. In fact, the key point can be any point ranging from P_1 to P_{n-2}. Only after we traverse all possible values of the key point, can we find out the global optimal solution. The following theorem will illustrate it.

Theorem 1: the length of the optimal bitonic TSP loop consisting n points on a plane can be:

When n=1, $M[n]$=0

When n=2, $M[n]$=2 × D[1,2]

When n≥3, $M[n]$=

$$\min_{1\le k\le n-2}\{D[k,n]+\sum_{i=k+1}^{n-1} D[i,i+1]+M[k+1]-D[k,k+1]\}$$

Proof: when n=1 or n=2, the theorem is obviously true. When n≥3, the theorem has traversed all the possible values and worked out a series of local optimal loop. And we choose the shortest one to be the global optimal loop.

Let the key point of global optimal loop is P_{k1}. Assume that there is another loop that is shorter than the optimal loop. And in this loop, there will be a key point connecting P_n and we name it P_{k2}.

If $k1$=$k2$, according to lemma 1, any loop with the key point of P_{k1} cannot be shorter than the optimal loop, in which P_{k1} is the key point. So the shorter loop doesn't exist.

If $k1$≠$k2$, according to lemma 1, any loop with the key point of P_{k2} cannot be shorter than the optimal loop with the key point of P_{k2}. However, the optimal loop determined by the key point P_{k1} is the shortest among other optimal loops determined by all the other key points. All the same, this kind of loop doesn't exist either.

So far, the proof of theorem 1 is accomplished.

IV. Realization and Optimization

A. Optimal Value Algorithm

Before working out the optimal solution by dynamic programming algorithm, we usually need to find the optimal value of the question firstly [8]. For the bitonic TSP question, when input set of points' coordinates on a plane, we use the pseudo-code of the optimal algorithm as followed:

```
OptValue (P[1…n], n)
    Compose P[1…n] according to X axis
    M[1]←0
    M[2]←2×D[1, 2]
    K[2]←1
    for i←3 to n do
        M[i] ← ∞
        link←0
        for k←i-2 down to 1 do
            link ←link +D[k+1, k+2]
            locallen←link +D[k, i]+M[k+1]–D[k, k+1]
            if locallen<M[i] then
                M[i] ←locallen
                K[i] ←k
    return K[1…n] and M[n]
```

In the algorithm above, $P[1…n]$ indicates the inputting set of points on a plane; *link* indicates the length of the up-

link of the right-half loop; *locallen* indicates the length of local optimal loop. $K[i]$ is used to store the key points of the optimal loops constructed by the first i points. The main structure of the algorithm consists two dual recursion, of which the time complexity is $O(n^2)$. As for the consumption of space, $M[1\ldots n]$ and $K[1\ldots n]$ will be stored in a 1-dimension array. It seems that a 2-dimension array is needed to store $D[i,j]$, the distance of two points. However it is not necessary because we can work out the $D[i,j]$ before using it, so the space complexity will be reduced to $O(n)$. We can use inline function Distance(i, j) to replace $D[i,j]$ and it can be accomplished within invariable time. Though in the classic algorithm, we can also use this method to achieve $D[i,j]$, the mid-result M[i,j] must be stored in a 2-dimension array.

Besides, this algorithm could be optimized farther. Within the internal recursion of the algorithm, most time are spent on calculating three times of the distance of two points, in which two times are calculating the distance of nearing points', namely $D[k,k+1]$ and $D[k+1,k+2]$. And it will lead to repeating calculation. To solve this problem, we can first work out the nearing points' distance and store them in a one-dimension array, so that two times of calculation could be saved in internal calculation.

B. *Optimal Solution Algorithm*

From the algorithm above, we can get $K[1\ldots n]$ and deduce the optimal path. The pseudo-code of the optimal algorithm is as followed.

```
OptSolution (P[1…n],K[1…n],n)
    Line(n-1,n)
    do
        k←K[n]
        Line (k, n)
        for i←k+1 to n-2 do
            Line (i, i+1)
        n←k+1
    while k!=1
    Line(1, 2)
```

In the optimal algorithm above, function Line(i,j) links P_i and P_j by a straight line. Finally, we can draw a complete optimal bitonic TSP loop and the algorithm can be accomplished in linear time.

V. TEST AND ANALYSIS

The author of this paper has conducted a test of the improved algorithm comparing with the classic. The hardware environment is this: P4 2.8G (dual core) CPU; 512MB memory; Windows XP operating system; VC++ 6.0 integrated developing environment; experiment data is stochastic set of points on a plane. Fig. 2 shows the executing result of the testing application.

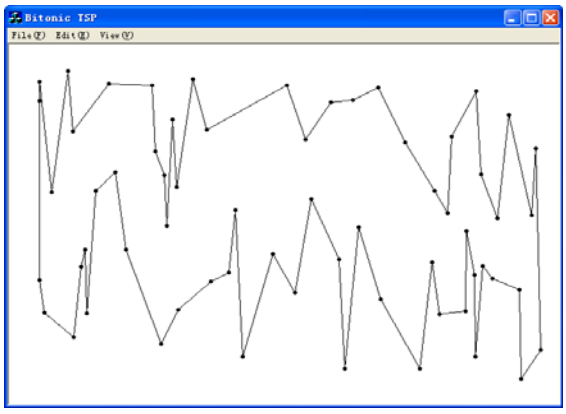

Figure 2. Executing result of testing application

For a more distinct comparison, we choose a comparable large scale input to conduct the experiment and separately record the result data of 1000 to 6000 points (shown as Fig. 3 and Fig. 4). The consumption of the memory shows the peak value of the memory being used when the program is executing one of the two algorithms. The time consumption shows the time needed to completely finish an algorithm.

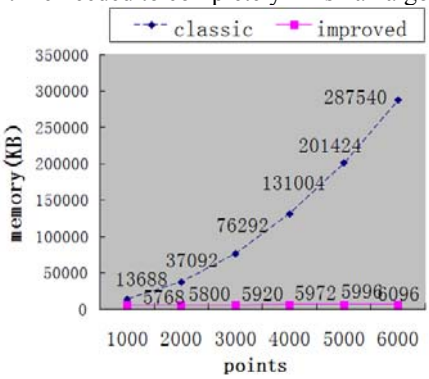

Figure 3. Comparison of memory consumption

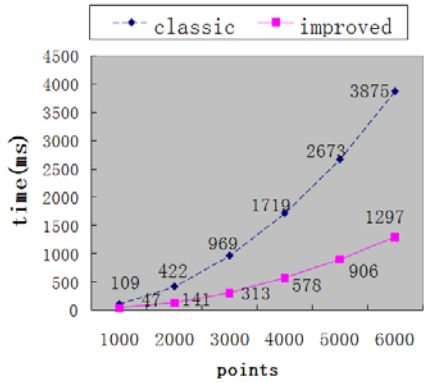

Figure 4. Comparison of time consumption

During the whole experiment process, with the same input, the results of the two algorithms are completely consistent, which farther proves the correctness of the improved algorithm. Classic algorithm's space requirement increases as a parabola when the inputting scale increases. If

Published by Atlantis Press, Paris, France.

the number of the inputting points is over 6000, then the physical memory will appear deficient and the operating system will distribute visual memory; and the calculation cannot continue if the inputting scale continues increasing. However, the improved algorithm only needs linear space. Compared with the classic algorithm, its increasing trend is not so obvious that the scale of problem the improved algorithm can solve augments largely. The time complexities of the two algorithms both are $O(n^2)$, but the improved algorithm is much more concise and efficient. Through optimizing, a lot of repeating calculations are reduced and the improved algorithm's computing speed has improved 2-3 times of the classic algorithm.

VI. EPILOGUE

The theory and the experiment both prove that the improved dynamic programming algorithm for bitonic TSP, this paper has put forward, is definitely correct. Compared with the classic algorithm, the improved algorithm analyzes the question from a different aspect and constructs a new optimal substructure, enabling it to reduce the space complexity to $O(n)$ while the time complexity stays the same. The experiment indicates that the space complexity also matters a lot on processing scale. Moreover, the improved algorithm's time efficiency has improved much over the classic algorithm because it is concise and easier to realize. In a word, even though the classic algorithm is widely spread, it may be not the optimal and also can be improved.

REFERENCES

[1] SHEN Hong-lian, ZHANG Guo-li, LI Zhen- tao. New algorithm for solving TSP. Computer Engineering and Applications,2008, 44(6) : 65- 67.

[2] Hassin R, Rubinstein S. Better approximations for max TSP[J]. Inf. Proc. Lett, 2000, 75(4): 181-186.

[3] Bentley J L.Fast algorithms for geometric traveling saleman problems [J]. ORSA Journal on Computing, 1992, 4(4): 387-411.

[4] WANG Xiao-dong. Design and analysis of computer algorithms [M].Beijing: Publishing House of Electronics Industry, 2007.

[5] ZHONG Xue-ling. Batch scheduling algorithm based on dynamic programming. Computer Engineering and Applications，2010，46（7）：229-231.

[6] Cormen T, Leiserson C, Rivest R.Introduction to algorithms [M]. Cambridge: MIT press, 1990.

[7] SUN Xiao-yan, LI Zi-liang, PENG Xiong-feng, FU Ya-li, LIANG Zhi-qiang: The dynamic programming applied to solving the shortest-path of transportation problem[J] Machinery Design and Manufacture, 2010(2): 223-224.

[8] LIU Jin-yi. An Improved Algorithm for t he Linear Partition Problem[J]Journal of Liaoning university of petrol EUM and chemical technology, 2007,27(3): 49-52.

Published by Atlantis Press, Paris, France.