A Novel Tour Construction Heuristic for Traveling Salesman Problem Using LFF Principle*

Sheqin Dong   Fan Guo   Jun Yuan   Rensheng Wang   Xianlong Hong

Department of Computer Science & Technology, Tsinghua University,   Beijing, 100084

Abstract

Less Flexibility First(LFF) principle is inspired and developed by enhancing some rule-of-thumb guidelines resulting from the generation-long work experience of human professionals in ancient days. In this paper, we generalize this principle to the classical Traveling Salesman Problem and propose a tour construction heuristic. Our new heuristic is closely related to Cheapest Insertion, a well-known heuristic for TSP. Then we prove that our algorithm can be implemented to run in O(n^4) time, achieving a tour no worse than Convex Hull, Cheapest Insertion (CHCI). Experimental results are comparable to simple local search heuristics such as 3-opt and baseline simulated annealing and better than any conventional tour construction heuristic at the expense of increased running time.

Keywords

Less Flexibility First, Traveling Salesman Problem, Tour Construction Heuristic

1. Introduction

The Traveling Salesman Problem (TSP) is known to be NP-hard [1]. Assuming the widely believed conjecture P≠NP, NP-hard problems cannot be solved to optimality within polynomially bounded computational time. Therefore, there is much interest in approximation algorithms that can only find near-optimal tours but do so quickly. Two broad classes of the approximation algorithm are constructive and local search methods. For the latter kind, 2-opt[2] and 3-opt[3] are the most simple yet quite efficient.

Tour construction heuristics are also known as successive augmentation heuristics. Typical examples are Nearest Neighbor, Greedy, Clarke-Wright Saving Heuristic [2] and Christofides [1]. Such heuristics build a tour from scratch by a growth process (usually a greedy one) until a feasible tour is constructed, while local search approaches start from a legal tour and try to improve its quality through neighborhood search. Tour construction heuristics not only serve as plausible mechanisms for generating initial tours needed by local search algorithms, they also provide a unique perspective from a theoretical point of view [1][4][5] and good results in practice. Whereas the constructive approach performs poorly on many combinatorial optimization problems, in the case of TSP, there are many algorithms that get within 15% of optimal and pay a relatively little price in running time[6].

In this paper, we adopt and generalize Less Flexibility First (LFF) principle first introduced in solving the block placement and rectangle packing problems in automated VLSI design [7]. We develop the concept of flexibility in the context of TSP. We will see how well this heuristic can currently provide by numerical experiment, in which we propose an efficient implementation as a worthwhile compromise between tour quality and running time cost.

The remainder of the paper is organized as follows. In Section 2, LFF algorithm on TSP is given. Section 3 demonstrates the time complexity and tour quality of our heuristic. Implementation issues and Experimental results are given in Section 4. Section 5 is conclusion.

2. LFF Algorithm

2.1 Flexibility in TSP

Similar to the procedure of packing objects in a work space[7], our heuristic for solving TSP belongs to the category of successive augmentation, known as tour construction heuristic for TSP. Also, we have to decide which city to insert and where (between which two adjacent cities in the partial tour) it should be inserted. The second question of finding an optimal inserting position is relatively easy: the inserting position is chosen to minimize the increase of tour length after the insertion of a city. So we have the following definition:

Definition 1 (Optimal inserting position)

Suppose a city cj is to be inserted to the current partial tour $T_k : \{c_{i_1}, c_{i_2}, \ldots, c_{i_k}, c_{i_1}\}$, and the distance matrix is dij(n*n), while dij is the distance from city ci to city cj. The optimal inserting position p is

$$\arg\min_{p \in \{1,\ldots,k\}} \{d_{i_p j} + d_{j i_{p+1}} - d_{i_p i_{p+1}}\}$$

City i_{k+1} denotes city i_1 in the above definition.

Unlike the rectangle packing problem, a city in TSP does not have a corresponding size or shape. And for a Euclidean TSP instance, the only property of a city is its position (two coordinates) in the plane, and this

*This work is supported by NSFC 60473126

property should be used to derive the concept of flexibility of a city. Since the optimal inserting position is already given, we look into the relative position of a candidate city and the two adjacent cities at its potential inserting position. An extreme case is that the candidate city is exactly on the line between the other two cities. Then the city is "bounded" by the other two and has no flexibility at all. If the city is quite close to the line, it has little flexibility because the insertion is almost necessary to achieve an optimal tour. On the other hand, if the city is farther, it has more freedom and a late insertion may be a better idea, accordingly the city gains more flexibility. (See Fig. 2).

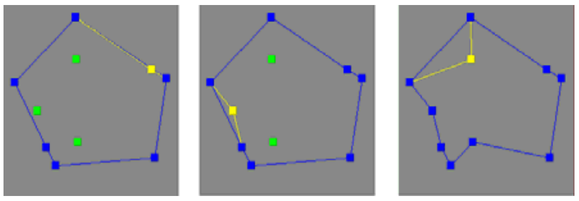
Figure 2. Cities with different flexibility

Definition 2 (Flexibility of a city)

Suppose current partial tour is $T_k : \{c_{i_1}, c_{i_2}, ..., c_{i_k}, c_{i_1}\}$, the optimal inserting position for a candidate city a city cj ($c_j \notin T_k$) is p ($1 \le p \le k$), then the flexibility of city c_j with respect to T_k is

$$flex(c_j, T_k) = \min_{q \in \{1,...,k\}} \left\{ d_{i_q j} + d_{j i_{q+1}} - d_{i_q i_{q+1}} \right\}$$
$$= d_{i_p j} + d_{j i_{p+1}} - d_{i_p i_{p+1}}$$

City i_{k+1} denotes city i_1 in the above definition. Note that since the triangular inequality holds for Euclidean TSP, the flexibility value is non-negative, and a smaller value shows that the candidate city is closer to the line between the other two. Although we derive this definition for Euclidean TSP, it can be applied to general TSP since only distance matrix is sufficient to make up the definition.

2.2 LFF in TSP

Here we define a city inserting move (CIM) as an augmentation step of the current partial tour such that the candidate city must be inserted at the optimal inserting position. An LFF city inserting move (LFFM) is a particular kind of CIM defined below:

Definition 3(CIM and LFFM in TSP)

Suppose current partial tour is $T_k : \{c_{i_1}, c_{i_2}, ..., c_{i_k}, c_{i_1}\}$, an LFFM in Traveling Salesman Problem is a pair <c_j, p> , where $c_j (c_j \notin T_k)$ is the candidate city to be inserted and $p (1 \le p \le k)$ is the inserting position. The pair satisfies the following conditions:

$$flex(c_j, T_k) = \{ d_{i_p j} + d_{j i_{p+1}} - d_{i_p i_{p+1}} \} \quad (1)$$
$$\min_{c_l \notin T_k} \{ flex(c_l, T_k) \} = flex(c_j, T_k) \quad (2)$$

City i_{k+1} denotes city i_1 in the above equations. Equation (1) ensures that p is the optimal inserting position for c_j and T_k. Equation (2) guarantees that city c_j has less flexibility than any other candidate cities. If only Equation (1) is satisfied, the pair is a CIM.

The heuristic in TSP based on LFF principle is similar to that in rectangle packing. It is implemented by the next two algorithms. The CIM performed in algorithm FFV below is tentative so that we can recover the tour after each evaluation.

Algorithm FFV: Fitness cost Function Value

```
Input: a city $c_j$, a partial tour T
Output: fitness value of $c_j$
Begin
    1. Perform a CIM inserting $c_j$ to T;
    2. Perform an LFFM inserting a city not included
       in the current partial tour;
    3. Goto step 2 until no cities left;
    4. Return the fitness cost function value.
End
```

Algorithm LFF: LFF construction heuristic

```
Input: a TSP instance
Output: a legal tour T
Begin
    1. Generate a partial tour $T_l$, set current partial tour
       T=$T_l$;
    2. For each city $c_j \notin T$, calculate FFV($c_j$, T);
    3. Insert the city with least FFV and update the
       current partial tour T;
    4. Goto step 2 until no cities left;
    5. Output T.
End
```

3. Analysis of LFF Heuristic

3.1 Tour quality

We have shown that the proposed LFF heuristic is closely related to Convex Hull, Cheapest Insertion (CHCI) heuristic. Here we demonstrate formally that LFF always achieves better or equal final tour quality. So we have an upper-bound both in practice and in worst-case.

Denote the partial tour that we get after inserting a city in Algorithm LFF by T_k, while the subscript k equals the number of cities in the partial tour. After a city c_{k+1} is inserted, T_k becomes T_{k+1}, then we have

Theorem 1: FFV(c_{k+1},T_k) $\le$ FFV(c_k,T_{k-1}). (3)

FFV(c_k,T_{k-1}) predicts the tour length we can obtain using Cheapest Insertion on T_k. Theorem 1 demonstrates that it is a non-increasing sequence as the parameter k increases. We can use Theorem 1 to derive two significant results on the tour quality of LFF tour construction heuristic.

Corollary 1 Starting with the same partial tour, LFF heuristic always reaches a tour no worse than Cheapest Insertion.

Corollary 2 Starting with the tour created from the convex hull, given a Euclidean TSP instance I, Algorithm LFF guarantees that $LFF(I)/OPT(I) \leq 3-2/N$. N is the number of cities in I.

3.2 Complexity of computational time

Lemma 1 The Cheapest Insertion heuristic can be implemented in $O(n^2)$ time.

Corollary 3 The computational complexity of Algorithm FFV is $O(n^2)$.

Theorem 2 The computational complexity of LFF construction heuristic is $O(n^4)$.

4. Implementation and Experiment

4.1 Implementation of LFF heuristic

LFF algorithm in Section 2.3 always evaluates the fitness cost function value(FFV) of every non-tour city. This is not efficient because only the city with least FFV need to be identified. Alternatively, we can evaluate the FFV of a small part of non-tour cities which are "most competitive candidates" to find the one with least FFV (or at least a good approximation).

A simple idea of which non-tour cities should be evaluated is to find the convex hull of non-tour cities. However, the loss of tour quality in this single convex hull approach is significant in practice and Theorem 1 in Section 3.1 no long holds for this case. It will hold if the city u in Section 3.1, and we call it here "Reference City", is included. (<u,p> is the first LFFM when the FFV of "Last City" is calculated in the previous iteration of LFF algorithm). So a better idea is to continuously perform the convex hull operation until "Reference City" is included. In Figure 5, only one operation is enough to reach the reference city; for a much larger problem size, multiple operations are needed.

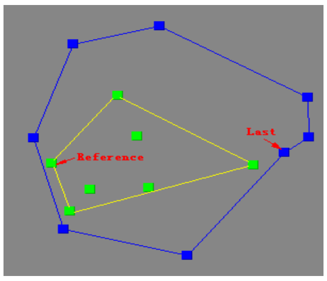

Figure 5. Convex Hull of Non-tour Cities and Reference City

Therefore, the previous time-saving technique can be applied when there are a large number of non-tour cities left rather than the whole procedure. It leads to a better tour at the expense of a slight increase in computational time. We set the threshold for the number of non-tour city to be max(C, αN). C and α are constants and N is the problem size. Our implementation choose C=100 and α=0.4.

4.2 Tour length and running time for LFF

In the experiment, we demonstrate performances of both the original LFF algorithm(LFF) and the variant proposed in Section 4.1(LFFR) for random Euclidean TSP instances and those from TSPLIB95[9]. Two conventional tour construction heuristic: Nearest Neighbor(NN) and Convex Hull Cheapest Insertion(CHCI) are also tested so that closeness to optimality and the computational time can be compared. The results are listed in Table 1.

Table 1. Tour quality for tour construction heuristics

Average Percent Excess over optimality					
Random Euclidean Instances					
N =	100	200	300	400	500
NN	23.8	23.5	22.0	24.6	24.0
CHCI	8.01	11.3	13.1	14.1	14.7
LFF	1.74	1.98	2.38	-	-
LFFR	1.74	2.62	3.55	3.96	4.26
TSPLIB95 Euclidean Instances					
N =	100-200	200-300	300-400	400-500	500-1k
NN	21.1	21	28.0	22.5	28.8
CHCI	6.47	11	11.6	10.8	13.2
LFF	1.24	2.33	2.2	-	-
LFFR	1.23	2.79	2.25	2.44	3.1

All computations of the algorithm written in C++ are performed on an IBM PC(Pentium III 1.13GHz, 384MB main memory). Random instances are generated using the "portgen" utility downloadable from SGI challenge page[3].

Our LFF heuristic obtains much better solutions than conventional tour construction heuristics. For TSPLIB instances, LFFR is approximately 9 times closer to optimality than NN, and 4 times than CHCI. The max excess of LFFR is 4.75% and the average is 2.00%. For NN and CHCI, the min excesses are 5.41% and 1.95% each. LFF obtains slightly better tours than LFFR, but the running times are 1 to 4 times longer. For the two well-known TSPLIB instances pcb442 and att532, LFFR obtains 52242(2.88%) and 28369(2.47%) tour length.

For random generated instances, the final tour quality degrades more for LFFR than LFF when the problem size N increases; yet LFFR runs much faster. The comparison of tour quality between LFF heuristic and NN or CHCI is similar to TSPLIB instances. The computational time for both LFF and LFFR grows at the rate of N^4. Table 2 and Table 3 summarize the tour length and running time

Table 2. Tour quality and Running Times for heuristics

Random Euclidean Instances				
	Average Percent Excess		Running Time in Seconds	
Algorithms	$10^{2.0}$	$10^{2.5}$	$10^{2.0}$	$10^{2.5}$
LFF	1.74	2.3	2.16	239
LFFR	1.74	3.46	2.16	64.7
CHR	9.5	9.9	0.01	0.04
2-Opt	4.5	4.8	0.01	0.03
3-Opt	2.5	2.5	0.01	0.04

Table 3. Tour quality and Running Times for heuristics

Random Euclidean Instances				
	Average Percent Excess		Running Time in Seconds	
Algorithms	$10^{2.0}$	$10^{2.5}$	$10^{2.0}$	$10^{2.5}$
LFF	1.74	2.3	2.16	239
LFFR	1.74	3.46	2.16	64.7
SA_1 α=1	3.4	3.7	3.88	58.8
SA_1P α=10	1.7	1.9	10.0	48.4
SA_2 α=100	1.1	1.3	44.1	205

obtained by different algorithms. The average percent excess is measured over Held-Karp lower bound. Running Time in Seconds is normalized to be the predicted user seconds on Compaq ES40 6/500 alpha machine, 500 Mhz Processor, 2Gb of RAM. In table 3, CHR is short for Christofides, which provides a better worst-case guarantee than any other tour construction heuristic and find better tours in practice. In table 4, SA_1 is short for baseline simulated annealing, SA_1P is SA_1 with neighborhood pruning. SA_2 is SA_1 with neighborhood pruning and low temperature start. α is the coefficient of temperature length. Results of algorithms other than LFF or LFFR are from [6]. The normalization method for running times can be found in 8th DIMACS Implementation Challenge Page. [3]

From Table 3, we observe that LFF and LFFR can obtain better tours than Christofides and 2-Opt. The tour qualify of LFFR is comparable to 3-Opt. The running times for LFF heuristics is still thousands of times longer than other heuristics.

We can see in Table 4 that LFF and LFFR performs better than SA_1 with α=1 and SA_1P with α=10 on instances of size 10^2. The performance of LFFR is comparable to SA_1 with α=1 on instances of size $10^{2.5}$, but is worse than SA_1P with α=10.

5. Conclusion

We have generalized the less flexibility first principle to traveling salesman problem and propose a deterministic heuristic that can find a better tour than any conventional tour construction heuristic at the expense of relatively high running time. The procedure of our heuristic has an iterative improvement property which is similar to local search heuristics, and the tour quality obtained is also comparable to 3-Opt. The performance is better than baseline SA and dominated by SA_2 with some speed-up techniques. Using k-d tree rather than the link structure in our implementation of LFF heuristic is expected to reduce the running time significantly. Also, the final tours are typically not 2-optimal, so adding a post-processing 2-Opt phase will lead to a better tour quality with an insignificant increase in total running time.

6. References

[1] N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, GSIA, Carnegie-Mellon University, 1976.

[2] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568-581, 1964.

[3] D. S. Johnson. *8th DIMACS Implementation Challenge: The Traveling Salesman Problem*. http://www.research.att.com/~dsj/chtsp/index.html. Latest Update: 16 November 2004.

[4] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245-2269, 1965.

[5] G. Reinelt. *The Traveling Salesman Problem: Computational Solutions for TSP Applications*, Lecture Notes in Computer Science 840, Springer-Verlag, 1994.

[6] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215-310, John Wiley & Sons, New York, 1997.

[7] S. Dong, Y. Lin, X. Hong, Y. L. Wu, and J. Gu. VLSI block placement using Less Flexibility First principles. In *Proc. ASPDAC'01*, pages 601-604, Japan, 2001.

[8] R. L. Graham, An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132-133, 1972.

[9] H. L. Ong and J. B. Moore, Worst-case analysis of two travelling salesman heuristics. *Operations Research Letters*, 2:273-277, 1984.