An Iterated Local Search Approach for Finding Provably Good Solutions for Very Large TSP Instances

Peter Merz[1,⋆] and Jutta Huhse[2]

[1] Department of Computer Science
University of Kaiserslautern, Germany
peter.merz@ieee.org
[2] IBFI Schloss Dagstuhl
Wadern, Germany
jutta.huhse@dagstuhl.de

Abstract. Meta-heuristics usually lack any kind of performance guarantee and therefore one cannot be certain whether the resulting solutions are (near) optimum solutions or not without relying on additional algorithms for providing lower bounds (in case of minimization).

In this paper, we present a highly effective hybrid evolutionary local search algorithm based on the iterated Lin-Kernighan heuristic combined with a lower bound heuristic utilizing 1-trees. Since both upper and lower bounds are improved over time, the gap between the two bounds is minimized by means of effective heuristics. In experiments, we show that the proposed approach is capable of finding short tours with a gap of 0.8% or less for TSP instances up to 10 million cities. Hence, to the best of our knowledge, we present the first evolutionary algorithm and meta-heuristic in general that delivers provably good solutions and is highly scalable with the problem size. We show that our approach outperforms all existing heuristics for very large TSP instances.

1 Introduction

The Traveling Salesman Problem (TSP) is one of the best-known combinatorial optimization problems. Simply stated, the problem is to find the shortest round trip through a set of cities where each city has to be visited exactly once. Unfortunately, the TSP is known to be NP-hard.Meta-heuristics usually do not provide a performance guarantee such as approximation algorithms. Hence, for finding provably good solutions, one has to resort to exact algorithms like Branch & Cut, or use an algorithm for computing a lower bound (in case of a minimization problem) additionally to the meta-heuristic.

In this paper, we present a heuristic approach that simultaneously improves lower and upper bounds for a TSP instance to provide a gap for the best solution found. The gap determines the maximum deviation from the optimum solution

⋆ This work is partially supported by the German Research Foundation (DFG).

and therefore provides a quality measure for the obtained TSP tour. The approach differs from exact algorithms like Branch & Cut [1] in that no efficient linear programming (LP) solver is required and it differs from approximation algorithms such as PTAS [2] in that no general performance guarantee is provided. Instead, the quality is proved for each instance and a particular run: a final gap between lower and upper bound of 1% means that the solution found is at most one percent above the optimum (in practice the real gap is much lower).

We show in experiments that our approach (a) delivers solutions known to be only about 1% above the optimum on average, (b) scales linearly for random Euclidean instances and is therefore even applicable to instances with 10 million cities, and (c) outperforms all known TSP heuristics for very large instances.

The paper is organized as follows: Section 2 discusses state-of-the-art metaheuristics for the TSP. In Section 3, our new highly effective approach is presented. Results from several experiments are discussed in Section 4. The paper is concluded in Section 5.

2 Effective Approaches for the TSP

The TSP has served as a test-bed for new heuristic approaches including evolutionary algorithms (EA). Consequently, many approaches, both evolutionary and non-evolutionary, have been proposed. Here, we focus on those approaches which are highly effective and scalable. TSP instances up to a size of 1,000 can be considered as trivial for most algorithms. In fact, these small problems can usually be solved exactly by Branch & Cut [3] in a few seconds. Therefore, these instances are no longer of interest for heuristics research on the TSP. For instances up to approx. 30,000 cities, very effective heuristics have been proposed most of which are based on the powerful Lin-Kernighan (LK) heuristic [4], a variable k-opt local search. An example is Helsgaun's LK implementation (LKH) [5].

Only few evolutionary algorithms can compete with LKH. One of the best evolutionary approaches is the EA of Nagata using EAX crossover [6] and 2-opt local search. This algorithm finds (near) optimal tours up to a size of 33,000 cities, although with a high runtime. Recently, Nguyen *et al.*[7] have proposed a hybrid evolutionary algorithm which utilizes a variant of the MPX crossover operator [8] and a Lin-Kernighan local search variant with 5-opt moves. Results are reported for instances up to 85,900 cities. The authors claim that their algorithm is more effective than LKH. Moreover, the authors describe an approach for solving the World TSP (approx. 2 million cities) by solving and merging subproblems. But results for other instances in the range from 100,000 to 10 million cities are not reported.

For instances larger than 100,000 cities, only few heuristics have been proposed. For these instances, the DIMACS TSP implementation challenge [9] lists several approaches of which the best are based on the LK heuristic: The multi-level algorithm of Walshaw [10] first reduces the size of a TSP instance stepwise and then applies the (chained) LK heuristic to the smaller problems. The results are inferior to the results obtained by directly applied chained LK or iterated LK heuristics.

These heuristics are based on the principle of iterated local search [11], an evolutionary heuristic incorporating local search. The idea is to stepwise improve the current best solution by mutating it and subsequently applying local search. The first iterated local search was the iterated Lin-Kernighan (ILK) heuristic by Johnson [12]. Other variants have been proposed such as the chained Lin-Kernighan heuristic [13,14]. These ILK heuristics have been applied to instances with up to 10 million cities. The only algorithm within the DIMACS challenge not using LK as a subroutine and still being highly effective for large instances is the dynamic programming approach of Balas and Simonetti [15].

Except for the LKH heuristic, none of the mentioned algorithms provides a lower bound on the optimum solution. To the best of our knowledge, the only evolutionary algorithm computing lower bounds is the one proposed in [16]. However, the approach deals with instances below 2,400 cities only.

3 A Scalable Evolutionary Algorithm for the TSP

In the following, we present an ILK variant for the TSP that can be applied to instances with millions of cities.

3.1 The General Evolutionary Framework

The evolutionary framework we use in our algorithm is not specific to the TSP. The concept of iterated local search has been applied to other combinatorial problems with great success [11]. The framework is rather simple: First, a solution to the problem is generated using some sort of randomized construction heuristic. Then, a local search is applied to obtain a local optimum. Afterwards, the best local optimum obtained so far is mutated repeatedly by some problem–specific mutation operator, and a local search procedure is applied subsequently. If the newly obtained solution is better than the previous one, it is accepted as the new best solution. In this way, one can obtain successively better solutions. The reason why this approach is so effective for the TSP is that the fitness landscape of the TSP is highly correlated: The smaller the distance to the optimum, the better the fitness (the smaller the tour length in case of the TSP). Therefore, iterated local search allows to 'jump' from one local optimum to a better local optimum until the global optimum is reached. Relatively small mutations are necessary to jump to a new local optimum since local optima are close to each other [17,18].

Our local search is based on the LK heuristic, hence our iterated local search is called iterated LK. The general outline of our iterated LK is shown in Fig. 1. In contrast to other approaches, our ILK incorporates a lower bound computation. This computation is interleaved with the optimization algorithm as can be seen in the figure: every 400 iterations of the ILK, the lower bound is improved until there appears to be no more improvement possible (the lower bound computation has converged). The lower bound computation possibly modifies the candidate edge set, which is used by the local search to look for improving moves (edge exchanges).

3.2 Implementation Details of the ILK

To find initial solutions (Init() in the pseudo code), we use the Quick-Boruvka heuristic [9,14], and the initial candidate set (FindInitialCandidateSet(Instance) in the pseudo code) is based on a subgraph containing the two nearest neighbors for each quadrant of a city [14]. This candidate set has the property of being connected. The candidate set is computed using a k-d-tree data structure [19]. A small candidate set is essential for the scalability of the approach. Having eight neighbors on average appeared to be reasonable.

Due to the complexity of state-of-the-art implementations of ILK, it is not possible to describe all the aspects here in fully detail. A forthcoming technical report will cover all these aspects.

Mutation Operator. The mutation operator used in the algorithm is non-sequencial four exchange [4,20] using a random walk on the candidate set to find edges to be included in the tour. This operator has been proven to be very effective in conjunction with Lin-Kernighan local search [14]. Hence, in each mutation as few as four edges are exchanged: Edges (t_1, t_2), (t_3, t_4), (t_5, t_6), and (t_7, t_8) are replaced by edges (t_1, t_4), (t_2, t_3), (t_5, t_8), and (t_6, t_7). The random walk omn the candidate edge set assures that edges with a relatively small length instead of arbitrarily long edges are included. We experimented with several other mutation schemes. This one appeared to be the best. We found that a reasonable number of steps for the random walk is 150 independently of the problem size. So, we used this value in our algorithm.

Local Search Operator. As mentioned before, we use a variant of the original Lin-Kernighan heuristic for the local search. Compared to the original LK, we use 3-opt moves as submoves instead of 2-opt moves at all levels: edges (t_1, t_2),

```
function ILK−PM(Instance : TspInstance, MaxIter : Integer) : TspTour;
  begin
    C := FindInitialCandidateSet(Instance);
    Tour := Init ();
    Tour := LocalSearch(C, Tour);
    C := FindInitialLowerBound(C, TourLength(Tour));
    for iter := 1 to MaxIter do begin
      Tbest := Tour;
      Tour := Mutate(Tour);
      Tour := LocalSearch(Tour);
      if TourLength(Tour) < TourLength(Tbest) then Tbest := Tour;
      if (iter % 400) = 0 then C := UpdateLowerBound(C, TourLength(Tbest));
    end
    return Tbest;
  end
```

Fig. 1. The Evolutionary Local Search Algorithm

(t_3, t_4), and (t_5, t_6) are replaced by edges (t_2, t_3), (t_4, t_5), and (t_6, t_1) in a sub move. The next sub move will start by replacing the last new edge (t_6, t_1). We do not use backtracking which simplifies the implementation drastically without affecting the performance. In this aspect our implementation is similar to LKH. As in other LK implentations we make use of Bentley's don't look bits concept [21]. Moreover, we use two-level trees to represent tours [22].

3.3 The Lower Bound Computation

Held and Karp [23,24] proposed a method based on Lagrangian relaxiation to compute lower bounds for the TSP. It is based on computing 1-trees, i.e. minimum spanning trees with one additional edge (the second shortest edge of a leaf in the tree). The approach is to find a transformation given by a vector $\pi = (\pi_1, \ldots, \pi_N)$ that maximizes the lower bound w

$$w(\pi) = L(T_\pi) - 2\sum_{i=1}^{N} \pi_i, \tag{1}$$

where N is the problem size, T_π is a minimum 1-tree on the transformed graph G' for which the cost of traveling between city i and j is $c'_{ij} = c_{ij} + \pi_i + \pi_j$,

```
function UpdateLowerBound(C : CandidateSet; upper : REAL) :
TspTour;
  begin
    if (FirstTime) then begin
      InitPiValues(Pi);
      best_lower : = Calculate1Tree(Pi);
    end
    FirstTime := false;
    t := (upper − best_lower) / norm;
    for i := 1 to 200 do begin
      updatePi(Pi, t);
      lower : = Calculate1Tree(Pi);
      if (lower > best_lower) then begin
        best_lower := lower;
        Best_Pi := Pi;
        t := t ∗ 4.0;
      end
      t := t ∗ 0.75;
    end
    best_lower = Calculate1Tree(Best_Pi);
    return best_lower;
  end
```

Fig. 2. The Incremental Lower Bound Improvement

and $L(T_\pi)$ is the cost of the tree with respect to G'. Compared to other lower bounds this bound does not require to compute a linear program. In order to find the best π vector, a subgradient optimization can be applied. Within the subgradient optimization, π is updated as follows:

$$\pi_i = \pi_i + t^{(k)}\,(0.7(d_i^{(k)} - 2) + 0.3(d_i^{(k-1)} - 2)), \tag{2}$$

where $d_i^{(k)}$ denotes the degree of city i in the minimum 1-tree at step k. For $t^{(k)} \rightarrow 0$, for $k \rightarrow \infty$, and $\sum t^{(k)} = \infty$, $w(\pi)$ will converge to the maximum of $w(\pi)$.

However, in practice convergence can be very slow and since computing minimum 1-trees is expensive for very large instances, we use a simplified scheme of adjusting $t^{(k)}$ as shown in Fig. 2 and we repeat the subgradient optimization several times. In the figure, *norm* denotes $\sum_i (d_i^{(k)} - 2)^2$.

Since computing the minimum 1-tree for very large instances is time consuming, we calculate the minimum 1-tree in the candidate set. The final 1-tree calculation of each call to UpdateLowerBound() is computed after the candidate set was recomputed on the transformed instance using the best π vector. The ILK will use the recomputed candidate set in its subsequent iterations.

4 Experimental Evaluation

To assess the performance of our algorithm we performed several runs on a set of publicly available benchmark instances. We used all seven Euclidean TSPLIB instances of size >10,000, three national TSP instances of size >10,000, three VLSI instances of size >100,000 from `http://www.tsp.gatech.edu/`, and finally seven random Euklidean instances form the DIMACS TSP challenge in the range between 10 thousand and 10 million. We report average values of 32 runs for each instance. The algorithms were coded in C++ (under Linux with gcc) and running times are reported for an Intel Core Quad 6600 processor. For each run only one CPU core was used. Results are reported in Table 1. For each instance, the name (containing the size of the instance), the average final tour length, the standard deviation of the tour length (sdev), the percentage excess over the best–known solution (or in case of the E*.0 instances over the Held-Karp lower bound), the gap of the computed lower and upper bound, and the running time in seconds are provided. The termination criterion was $0.1N$ iterations (N is the problem size). The last two columns contain the average percentage excess and the running time for the alogrithm without lower bound computation.

The results demonstrate that the algorithm is capable of finding provably good solutions in very short time: For the TSPLIB instances, the gap lies between 0.64% and 1.51% by spending at most 40 seconds of CPU time. For the national instances, the gap is about 1.0% with a maximum of 122 seconds. The VLSI instances are significantly greater and the running time increases up to 1,192 seconds. The average gap is about 1.5%. Finally, for the random Euclidean instances the gap is below 0.83% independently of the size of the instance. The running time increases from 13 seconds (10 thousand cities) to 32,789 seconds

Table 1. Results of our ILK (ILK-PM-.1N) for DIMACS TSP Challenge Instances

	With Lower Bound					Without LB	
Instance	Tour length	sdev	Excess	Gap	sec	Excess	sec
rl11849	926071.6	583.5	0.302%	1.51%	9	0.382%	4
usa13509	20011164.1	6114.4	0.142%	0.91%	24	0.156%	18
brd14051	470051.6	230.9	0.148%	0.92%	20	0.140%	12
d15112	1574915.0	175.8	0.123%	0.67%	24	0.123%	16
d18512	646089.9	89.5	0.138%	0.64%	22	0.138%	13
pla33810	66456993.5	43691.9	0.618%	1.44%	20	0.693%	6
pla85900	143254481.1	85239.3	0.612%	1.24%	40	0.634%	13
sw24978	857594.8	257.0	0.234%	1.13%	33	0.262%	16
bm33708	961536.8	209.0	0.234%	0.98%	54	0.253%	17
ch71009	4573501.2	547.5	0.153%	0.86%	122	0.163%	70
sra104815	252310.2	47.7	0.377%	1.11%	152	0.443%	50
ara238025	581381.4	157.8	0.435%	1.32%	495	0.511%	121
lra498378	2183574.5	1389.0	0.705%	1.99%	722	0.886%	244
lrb744710	1619145.0	538.5	0.435%	1.36%	1192	0.541%	321
E10k.0	71969032.4	20938.0	0.850%	0.86%	13	0.853%	10
E31k.0	127469229.7	16091.7	0.786%	0.80%	50	0.792%	34
E100k.0	226146457.7	15767.1	0.809%	0.82%	184	0.819%	123
E316k.0	401954702.2	15174.6	0.801%	0.82%	665	0.808%	428
E1M.0	714351765.1	18622.6	0.797%	0.81%	2677	0.804%	1449
E3M.0	1269419125.3	14147.1	0.748%	0.81%	11547	0.755%	5228
E10M.0	2256845968.6	12030.8	0.752%	0.81%	32789	0.760%	17867

(10 million cities). Without the lower bound computation and the update of the candidate set the running time is considerably lower for the larger instances. Moreover, the average final tour quality is in all cases but one lower with candidate set update deactivated. The results are up to 0.18% better for the algorithm with lower bound computation. However, this appears to be dependent on the problem instance. For the uniform random instances, the gain is only about 0.01%. The runtime increases almost linearly with the problem size for our ILK as Fig. 3 demonstrates. In order to compare with other state-of-the-art approaches, Table 2 shows a comparison with the eleven best performing algorithms (out of 90) listed on the DIMACS TSP challenge web page. The summary was produced with the statistics code from the challenge. Thus the running time reported in the table is normalized to a DEC Alpha processor with 500 MHz in order to allow a comparison of the different approaches. The quality is given as the percentage excess over the Held-Karp (HK) bound. As shown in the table, our algorithm provides a significantly better tour quality than the other approaches. And it does this in a fraction of time of the second best approach which is also an ILK implementation. Note that none of the competitors

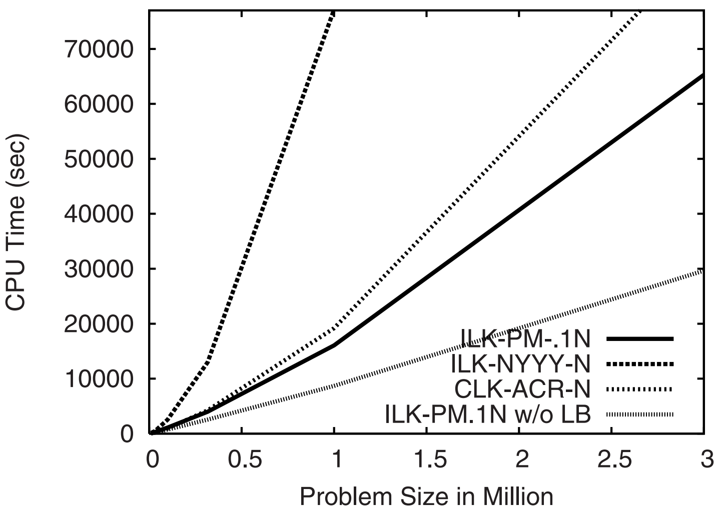

Fig. 3. Scaling behaviour of three ILK variants

computes a lower bound. For the 10 million city instance E10M.0, the quality of our approach is 0.75% over the Held-Karp bound compared to the best algorithm of the DIMACS challenge which is 1.63% over the Held-Karp bound!

Table 2. Comparison of DIMACS TSP Challenge Results on E1M.0. ILK-PM-.1N denotes our ILK with 1 million iterations and ILK-PM-.1N denotes our ILK with 1,2 million iterations.

% HK	Seconds	Implementation	Reference
0.787	17544.0	ILK-PM-.12N	this paper
0.792	77161.6	ILK-NYYY-N	([25])
0.797	16062.0	ILK-PM-.1N	this paper
0.804	8694.0	ILK-PM-.1N without LB	this paper
0.841	6334.0	ILK-NYYY-Ng	([25])
0.879	42242.5	MLCLK-N	[10]
0.888	3480.2	ILK-NYYY-.5Ng	([25])
0.903	19182.7	BSDP-6	[15]
0.903	19503.1	BSDP-8	[15]
0.903	21358.3	BSDP-10	[15]
0.903	19108.1	CLK-ABCC-N.Sparc	[13]
0.905	19192.3	CLK-ACR-N	[14]
0.910	16008.0	CLK-ABCC-N.MIPS	[13]
0.945	20907.6	MLCLK-.5N	[10]

This is due to the fact that the best algorithms for the smaller instances do not scale as well as our approach: Fig. 3 shows the normalized runtime of our approach (ILK-PM-.1 with and without lower bound computation), the ILK of Nguyen *et al.* (for which no reference exists except for the DIMACS challenge web page) denoted ILK-NYYY-N, and the runtime of chained LK of Applegate *et al.* denoted CLK-ACR-N [14] depending on the problem size. While the runtime of our approach without lower bound computation grows linearly with the problem size, the runtime of the others clearly grows faster and and yields in the non-applicability of these algorithms to very large problem instances (>1 million) whereas our approach is still very successful even if the lower bound computation is activated.

5 Conclusions

We presented a new Iterated Lin-Kernighan heuristic based on the powerful concept of iterated local search. We have shown in experiments that our approach scales well with the problem size and is therefore applicable to very large TSP instances with 10 million cities. Besides the scalability, the approach provides provably good solutions since it computes a lower bound interleaved with the optimization. Therefore, the obtained results are known to be not more than about 1% above the optimum and in case of the very large random Euclidean instances not more than 0.81% above the Held-Karp Lower bound even for the largest, 10 million cities instances. Compared to other evolutionary and non-evolutionary approaches for very large instances above 1 million cities, our approach obtains better tour quality in even shorter time. In particular, all algorithms from the DIMACS TSP implementation challenge are shown to be inferior to our approach. To the best of our knowledge the proposed approach is the only one that is both scalable to millions of cities and provides provably good solutions.

There are some issues for future research. Currently, we are working on a distributed algorithm for large instances based on the ILK presented here. Both the use of a population and the use of recombination are subject of our studies. Finally, we believe that our lower bound computation can be further improved.

References

1. Dantzig, G.B., Fulkerson, D.R., Johnson, S.M.: Solution of a Large-Scale Traveling Salesman Problem. Operations Research 2, 393–410 (1954)
2. Arora, S.: Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems. Journal of the ACM 45, 753–782 (1998)
3. Applegate, D., Bixby, R., Chvátal, V., Cook, B.: Finding Cuts in the TSP (A preliminary report). Technical Report 95-05, DIMACS (1995)
4. Lin, S., Kernighan, B.: An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operations Research 21, 498–516 (1973)

5. Helsgaun, K.: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. European Journal of Operational Research 126, 106–130 (2000)
6. Nagata, Y.: New EAX Crossover for Large TSP Instances. In: Runarsson, T.P., Beyer, H.G., Burke, E., Merelo-Guervos, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 372–381. Springer, Heidelberg (2006)
7. Nguyen, H.D., Yoshihara, I., Yamamori, K., Yasunaga, M.: Implementation of an Effective Hybrid GA for Large-Scale Traveling Salesman Problems. IEEE Transactions on Systems, Man and Cybernetics, Part B 37, 92–99 (2007)
8. Mühlenbein, H.: Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization. In: Schaffer, J.D. (ed.) Proceedings of the Third International Conference on Genetic Algorithms, pp. 416–421. Morgan Kaufmann, San Francisco (1989)
9. Johnson, D.S., McGeoch, L.A.: Experimental Analysis of Heuristics for the STSP. In: Gutin, G., Punnen, A. (eds.) The Traveling Salesman Problem and its Variations, Kluwer Academic Publishers, Dordrecht (2002)
10. Walshaw, C.: A Multilevel Approach to the Travelling Salesman Problem. Operations Research 50, 862–877 (2002)
11. Lourenco, H.R., Martin, O., Stützle, T.: Iterated Local Search. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics, Kluwer Academic Publishers, Dordrecht (2003)
12. Johnson, D.S.: Local Optimization and the Traveling Salesman Problem. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 446–461. Springer, Heidelberg (1990)
13. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: Finding Tours in the TSP. Technical Report Report Number 99885, Research Institute for Discrete Mathematics, University of Bonn, Germany (1999)
14. Applegate, D., Cook, W., Rohe, A.: Chained Lin-Kernighan for Large Traveling Salesman Problems. INFORMS Journal on Computing 15, 82–92 (2003)
15. Balas, E., Simonetti, N.: Linear Time Dynamic-Programming Algorithms for New Classes of Restricted TSPs: A Computational Study. INFORMS J. on Computing 13, 56–75 (2000)
16. Marinakis, Y., Migdalas, A., Pardalos, P.M.: A Hybrid Genetic–GRASP Algorithm Using Lagrangean Relaxation for the Traveling Salesman Problem. Journal of Combinatorial Optimization 10, 311–326 (2005)
17. Merz, P., Freisleben, B.: Fitness Landscapes and Memetic Algorithm Design. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 245–260. McGraw–Hill, London (1999)
18. Merz, P.: Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies. PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany (2000)
19. Bentley, J.L.: K-d-Trees for Semidynamic Point Sets. In: Proceedings of the Sixth Annual ACM Symposium on Computational Geometry, pp. 187–197 (1990)
20. Merz, P., Freisleben, B.: Memetic Algorithms for the Traveling Salesman Problem. Complex Systems 13, 297–345 (2001)
21. Bentley, J.L.: Experiments on Traveling Salesman Heuristics. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 91–99 (1990)
22. Fredman, M.L., Johnson, D.S., McGeoch, L.A., Ostheimer, G.: Data Structures for Traveling Salesmen. Journal of Algorithms 18, 432–479 (1995)

23. Held, M., Karp, R.M.: The Traveling-Salesman Problem and Minimum Spanning Trees. Operations Research 18, 1138–1162 (1970)
24. Held, M., Karp, R.M.: The Traveling-Salesman Problem and Minimum Spanning Trees: Part II. Mathematical Programming 1, 6–25 (1971)
25. Nguyen, H.D., Yoshihara, I., Yamamori, K., Yasunaga, M.: A New Three-Level Tree Data Structure for Representing TSP Tours in the Lin-Kernighan Heuristic. IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences E90-A, 2187–2193 (2007)