

# Finding Good Tours for Huge Euclidean TSP Instances by Iterative Backbone Contraction

Christian Ernst<sup>1,3</sup>, Changxing Dong<sup>1</sup>, Gerold Jäger<sup>1,2</sup>,  
Dirk Richter<sup>1</sup>, and Paul Molitor<sup>1</sup>

<sup>1</sup> Martin-Luther-University Halle-Wittenberg, D-06120 Halle (Saale), Germany

<sup>2</sup> Christian-Albrechts-University Kiel, D-24118 Kiel, Germany

<sup>3</sup> GISA GmbH, D-06112 Halle (Saale), Germany

**Abstract.** This paper presents an iterative, highly parallelizable approach to find good tours for very large instances of the Euclidian version of the well-known Traveling Salesman Problem (TSP). The basic idea of the approach consists of iteratively transforming the TSP instance to another one with smaller size by contracting pseudo backbone edges. The iteration is stopped, if the new TSP instance is small enough for directly applying an exact algorithm or an efficient TSP heuristic. The pseudo backbone edges of each iteration are computed by a window based technique in which the TSP instance is tiled in *non-disjoint* sub-instances. For each of these sub-instances a good tour is computed, independently of the other sub-instances. An edge which is contained in the computed tour of *every* sub-instance (of the current iteration) containing this edge is denoted to be a pseudo backbone edge. Paths of pseudo-backbone edges are contracted to single edges which are fixed during the subsequent process.

## 1 Introduction

The Traveling Salesman Problem (TSP) is a well known and intensively studied problem [1, 5, 10, 17] which plays a very important role in combinatorial optimization. It can be simply stated as follows. Given a set of cities and the distances between each pair of them, find a shortest cycle visiting each city exactly once. If the distance between two cities does not depend on the direction, the problem is called *symmetric*. The size of the problem instance is defined as the number  $n$  of cities. Formally, for a complete, undirected and weighted graph with  $n$  vertices, the problem consists of finding a minimal Hamiltonian cycle. In this paper we consider *Euclidean TSP* (ETSP) instances whose cities are embedded in the Euclidean plane<sup>4</sup>.

Although TSP is easy to understand, it is hard to solve, namely  $\mathcal{NP}$ -hard. We distinguish two classes of algorithms for the symmetric TSP, namely heuristics

---

<sup>4</sup> However, the ideas presented in this paper can be easily extended to the case in which the cities are specified by their latitude and longitude, treating the Earth as a ball (see [19]).

and exact algorithms. For the exact algorithms the program package *Concorde* [1, 20], which combines techniques of linear programming and branch-and-cut, is the currently leading code. Concorde has exactly solved many benchmark instances, the largest one has size 85,900 [2]. On the other hand, in the field of symmetric TSP heuristics, Helsgaun's code [6–8, 21] (LKH), which is an effective implementation of the Lin-Kernighan heuristic [11], is one of the best packages. Especially for the most yet not exactly solved TSP benchmark instances [14–16, 18, 19] this code found the currently best tours.

An interesting observation [13] is that tours with good quality are likely to share many edges. Dong et al. [4] exploited this observation by first computing a number of good tours of a given TSP instance by using several different heuristical approaches, collecting the edges which are contained in each of these (not necessarily optimal) tours, computing the maximal paths consisting of only these edges, and contracting these maximal paths to single edges which are kept fixed during the following process. By the contraction step, a new TSP instance with *smaller* size is created which can be attacked more effectively. For some TSP benchmark instances of the VLSI Data Set [18] with sizes up to 47,608, this approach found better tours than the best ones so far reported.

The idea of fixing edges and reducing chains of fixed edges to single edges is not new. It has already been presented by Walshaw in his multilevel version of Helgaun's LKH [12]. Walshaw's process of fixing edges however is rather naive as it only matches vertices with their nearest unmatched neighbours instead of using more sophisticated edge measures.

An alternative to the approach would be fixing *without* backbone contraction. Thus the search space is considerably cut, although the size of the problem is not reduced. This basic concept of edge fixing was already used by Lin, Kernighan [11] and is implemented in LKH. The main difference between edge fixing without backbone contraction and the approaches presented in [4, 12] is the reduction of the size by contracting. This reduction has great influence to the effectiveness of the approach. The reason is that all the edges incident to an inner vertex of the contracted paths do not appear in the new instance anymore. Another idea related to [4] is Cook and Seymour's tour merging algorithm [3], which merges a given set of starting tours to get an improved tour.

The bottleneck of the approach presented in [4] when applied to *huge* TSP instances is the computation of several good starting tours, i. e., tours of high quality, by using several different TSP methods. Using *different* TSP heuristics during the computation of the starting tours hopefully increases the probability that edges contained in each of the starting tours are edges which are also contained in optimal tours.

This paper focuses on TSP instances with very large sizes. Only a tiny part of the search space of such a huge TSP instance can be traversed in reasonable time. To overcome this problem, huge TSP instances are usually partitioned. In our new approach, which handles ETSP, this partitioning is done by moving a window frame across the bounding box of the vertices. The amount of the stepwise shift is chosen as a fraction  $1/s$  of the width (height) of the window frame

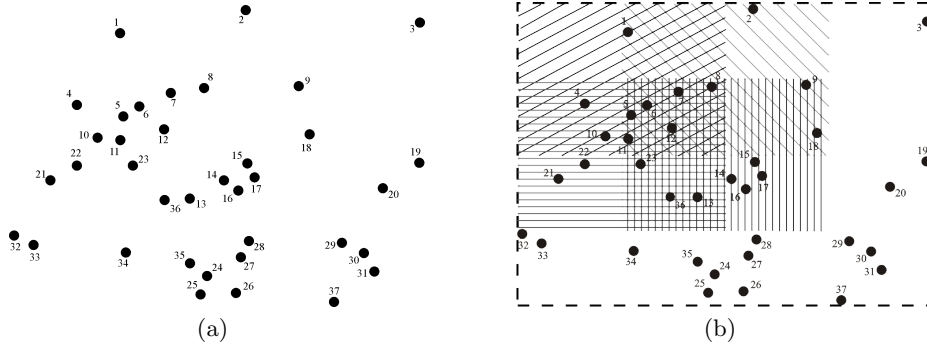
so that all vertices of the TSP instance but those located near the boundary are contained in exactly  $s^2$  windows (see Figure 1(b) where the basic idea is illustrated for  $s = 2$ ); parameter  $s$  determines the shift amount of the window frame. For the vertices of each window a good tour is computed by either an exact algorithm, e. g., Concorde, or some heuristics, e. g., by Helsgaun’s LKH. If two vertices  $u$  and  $w$  which are contained in the same  $s^2$  windows are neighbored in each of the  $s^2$  tours constructed, the edge  $\{u, w\}$  is assumed to have high probability to appear in an optimal tour of the original TSP instance, and we call it a *pseudo backbone edge*. As in [4, 12], maximal paths of pseudo backbone edges are computed and contracted to single edges which are fixed during the following process.

Our experiments show that **(a)** actually most of the fixed edges are contained in an optimal tour, and **(b)** fixing edges and contracting chains of fixed edges considerably reduce the size of the original TSP instance. Because of (b), the width and height of the window frame applied in the next iteration can be increased so that larger sections of the bounding box will be considered by each window. The iteration stops, when the window frame is as large as the bounding box itself. In this case, LKH is directly applied to the remaining TSP instance. The experimental runs show that tours of high quality of huge TSP instances are constructed by this approach in acceptable runtime. For instance, for `World-TSP` the approach computes a tour of length 7,525,520,531, which is only 0,18 % above a known lower bound, within 45 hours on a standard personal computer, and a tour of length 7,524,796,079, which is 0,17 % above the lower bound, within 13 hours on a parallel computer with 32 processors. Similarly, it finds a tour for the DIMACS 3,162,278-sized TSP instance [15], whose length is only 0,0465 % larger than the best tour currently known for that TSP instance, within 6 days, and a tour for the DIMACS 10,000,000-sized TSP instance, whose length is only 0,0541 % larger, within 20 days. Moreover, we observe a high-level trade-off between tour length and runtime which can be controlled by modifying, e. g., parameter  $s$  or some other parameters.

The paper is structured as follows. Basic definitions with respect to TSP and our approach are given in Section 2. The overall algorithm together with a detailed illustration is described in Section 3. The parameters of the algorithm are listed in Section 4. Section 5 presents the experimental results. Finally, conclusions and suggestions for future work are given in Section 6.

### Acknowledgement

The work presented in this paper was supported by the German Research Foundation (DFG) under grant MO 645/7-3.



**Fig. 1.** (a) Vertices embedded in the Euclidean plane with  $V = \{1, \dots, 37\}$ . (b) Four neighbored windows if the width (height) of the window frame is half the width (height) of the bounding box and parameter  $s$  is set to 2. The top left-hand (top middle-hand, middle left-hand, middle middle-hand) window is marked by slanted (back slanted, horizontal, vertical) lines. The vertices 5, 6, 7, 8, 11, and 12 are contained in each of the four windows.

## 2 Definitions

### 2.1 Basics

Let  $V$  be a set of  $n$  vertices embedded in the Euclidean plane (see Figure 1(a)) and let  $dist(u, w)$  be the Euclidean distance between the vertices  $u$  and  $w$ . A sequence  $p = (v_1, v_2, \dots, v_q)$  with  $\{v_1, v_2, \dots, v_q\} \subseteq V$  is called a *path of length  $q$* . The costs  $dist(p)$  of such a path  $p$  is given by the sum of the Euclidean distances between neighboring vertices, i. e.,  $dist(p) := \sum_{i=1}^{q-1} dist(v_i, v_{i+1})$ . The path is called *simple*, if it contains each vertex of  $V$  at most once, i. e.,  $v_i = v_j \Rightarrow i = j$  for  $1 \leq i \leq q$  and  $1 \leq j \leq q$ . It is called *complete*, if the path is simple and contains each vertex of  $V$  exactly once. It is called *closed*, if  $v_q = v_1$  holds. A complete path  $p = (v_1, v_2, \dots, v_n)$  can be extended to the closed path  $T = (v_1, v_2, \dots, v_n, v_1)$ . Such a closed path of  $V$  is called a *tour*.

### 2.2 Euclidean Traveling Salesman Problem

ETSP is the problem of finding a tour with minimum costs for a given set  $V$  of vertices embedded in the Euclidean plane. An ETSP instance is *constrained* by a set  $FE \subseteq \{\{v_i, v_j\}; v_i, v_j \in V \text{ and } v_i \neq v_j\}$  of *fixed edges*, if a tour  $T = (v_1, v_2, \dots, v_n, v_1)$  has to be computed such that

- (1)  $(\forall \{u, w\} \in FE)(\exists i \in \{1, \dots, n\}) \{v_i, v_{(i \bmod n)+1}\} = \{u, w\}$
- (2) there is no other tour  $T'$  which meets (1) such that  $dist(T') < dist(T)$ .

### 2.3 Contraction of a simple path

The basic step of our approach to compute good tours of very large ETSP instances is to contract a path to a single edge which is fixed during the subsequent iterative process. More formally, *contraction of a simple path*  $p = (v_i, v_{i+1}, \dots, v_{j-1}, v_j)$  transforms a constrained ETSP instance  $(V, FE)$  into a constrained ETSP instance  $(V', FE')$  with  $V' = V \setminus \{v_{i+1}, \dots, v_{j-1}\}$  and  $FE' = (FE \cup \{\{v_i, v_j\}\}) \cap (V' \times V')$ . Thus the inner vertices  $v_{i+1}, \dots, v_{j-1}$  of path  $p$  are deleted – only the boundary vertices  $v_i$  and  $v_j$  of  $p$  remain in the new constrained ETSP instance – and the size of the new constrained ETSP becomes smaller, unless  $i + 1 = j$ . (If  $i + 1 = j$ , the new instance is less complex than the current instance in the sense that the edge  $\{v_i, v_j\}$  is fixed.) Figures 4(b) and 4(c) illustrate the contraction process before and after the contraction of the four simple paths (5, 11), (6, 12, 7, 8), (9, 18), and (23, 36, 13).

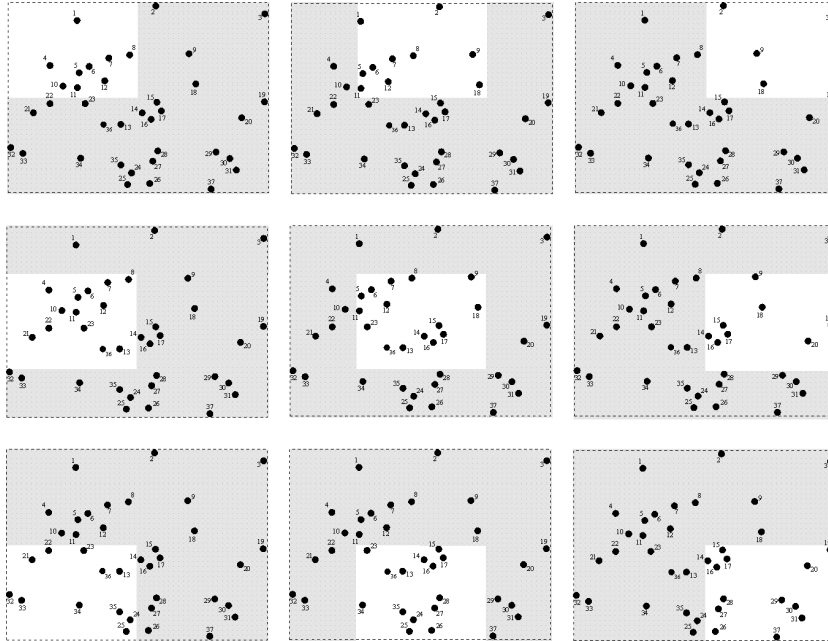
## 3 The Overall Algorithm

To solve very large (constrained) ETSP instances, we apply a window based approach to iteratively find paths to be contracted. In each iteration a window frame is moved across the bounding box of the vertices. As illustrated in Figure 2, the windows considered are *not* disjoint. In fact, we move the window by the fraction  $1/s$  of the width (height) of the window frame so that a vertex is contained in up to  $s^2$  windows. To simplify matters, we shall illustrate our approach with respect to  $s = 2$ , although  $s > 2$  might lead to better tours and actually does in some cases. The height and the width of the window frame are determined by dividing height and width of the bounding box by a parameter `WINDOW_SCALE` – you find more details on this parameter in Section 4 – which is chosen in such a way that the sub-problems induced by the windows have sizes which can be efficiently handled by Helsgaun’s LKH [6, 7, 21] or by the exact solver Concorde [20]. Since LKH finds optimal solutions frequently for small instances and the sizes of the windows are rather small, we have not tried to solve them with Concorde.

The sub-problems should contain a number of vertices greater than a lower bound `MNL` (see Section 4) so that a corresponding good tour contains ample information on the original TSP instance. In the following, we call a TSP instance containing less than `MNL` vertices a *trivial* instance. A tour is computed for each of the non-trivial sub-problems. Figure 3 illustrates this step which is applied in every iteration.

Now, our approach is based on the assumption, that two vertices  $u$  and  $w$  which are contained in the same  $s^2$  non-trivial windows and neighboring in each of the  $s^2$  tours constructed have high probability to appear in an optimal tour for the original ETSP instance<sup>5</sup> – for  $s = 2$ , in some sense, the four windows

<sup>5</sup> If one of the windows which contain  $u$  and  $w$  is trivial, the edge  $\{u, w\}$  is not considered as pseudo-backbone.

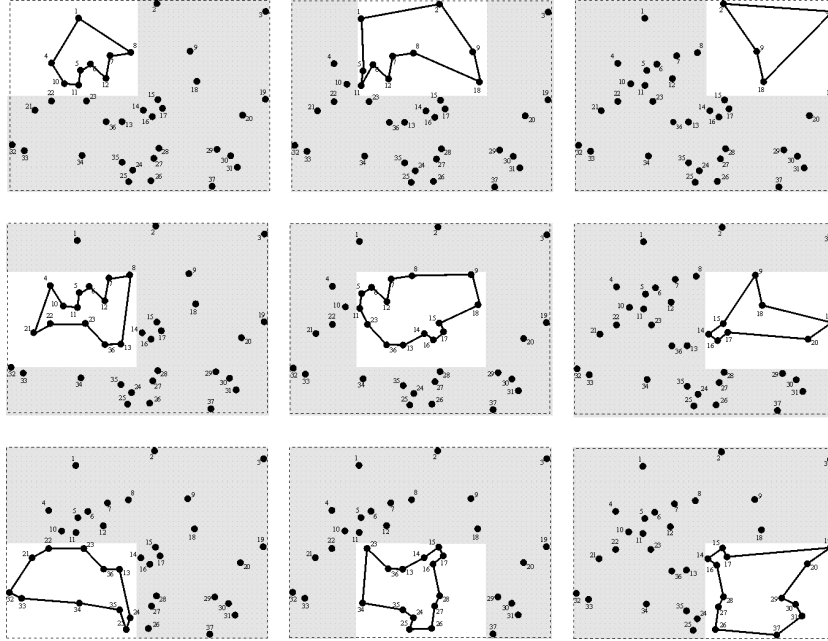


**Fig. 2.** Moving a window frame across the bounding box with `WINDOW_SCALE= 2` and  $s = 2$ . Each window defines a sub-problem for which a good tour is computed, independently of the neighboring sub-problems.

together reflect the surrounding area of the common vertices with respect to the four directions. We call such an edge  $\{u, w\}$  *pseudo backbone edge*.<sup>6</sup>

Figures 3 and 4(a) illustrate the notion of pseudo backbone edges. For this purpose, consider only the four top left-hand windows. Figure 3 shows a tour for each of these ETSP instances. Each of these four tours contain the edges  $\{5, 11\}$ ,  $\{6, 12\}$ ,  $\{7, 12\}$ , and  $\{7, 8\}$ . Thus, they are the pseudo backbone edges generated by these four tours. The set of all pseudo backbone edges generated by the tours computed in this iteration (Figure 3) is shown in Figure 4(b). The pseudo backbone edges can be partitioned into a set of maximal paths. In our running example the set consists of four paths, namely  $(5, 11)$ ,  $(6, 12, 7, 8)$ ,  $(23, 36, 13)$ , and  $(9, 18)$ . Now, these maximal paths are contracted, as described in Section 2.3 which leads to a new constrained ETSP with smaller or equal size (see Figure 4(c)).

<sup>6</sup> Note that edges at the boundary of the TSP instance cannot be contained in  $s^2$  non-trivial windows and therefore cannot be pseudo backbone edges. This may cause a problem if there are a lot of vertices at the boundary because, in this case, the original TSP instance cannot be reduced to such a degree that the final TSP instance is small enough to be efficiently solved by LKH. To overcome this problem, we shift the window frame above the boundary line.



**Fig. 3.** A good tour is computed for each of the non-trivial sub-problems. In this example, parameter `MNL` has been set to 3, so that there are no trivial instances.

This process is iteratively repeated. The parameter `WINDOW_SCALE` which determines the width and height of the window frame is re-adjusted, i. e., decreased, in each iteration. The iteration stops if `WINDOW_SCALE` is set to 1, i. e., if the window frame spans the whole bounding box. In this case, LKH is directly applied to the current constrained ETSP instance. Finally the fixed edges are recursively re-substituted which results in a tour of the original ETSP instance.

## 4 The Algorithm's Parameters

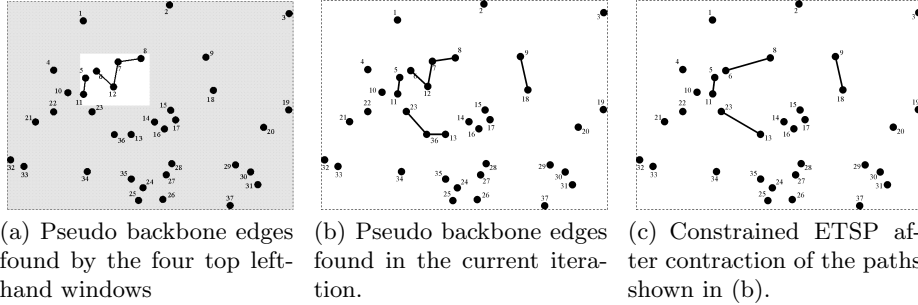
The main parameters of the algorithm are the following.

1. The scale parameter `IWS`<sup>7</sup> which determines the width and height of the window frame applied in the first iteration, namely

$$\text{width}_{\text{frame}} = \left\lceil \frac{\text{width}_{\text{bounding box}}}{\text{WINDOW\_SCALE}} \right\rceil \quad \text{and} \quad \text{height}_{\text{frame}} = \left\lceil \frac{\text{height}_{\text{bounding box}}}{\text{WINDOW\_SCALE}} \right\rceil$$

with `WINDOW_SCALE := IWS`.

<sup>7</sup> `IWS` is an abbreviation for `INITIAL_WINDOW_SCALE`



**Fig. 4.** Pseudo backbone contraction

2.  $D^8$  which is the factor relative to the width (height) of the window frame by which the window frame is shifted, i. e.,  $D = 1/s$ .
3. The minimum number  $MNL^9$  of vertices which a window has to contain so that a tour is computed for that sub-problem. If a window contains less vertices than  $MNL$ , no tour of the vertices contained in that window is computed (and no edge contained in that window becomes a pseudo backbone edge), as those tours would be suitable to only a limited extent for finding good pseudo backbone edges.
4. The parameter  $WGF^{10}$  which is the factor by which the window frame is re-adjusted after each iteration, i. e.,

$$WINDOW\_SCALE_{new} = \frac{WINDOW\_SCALE_{old}}{WGF}.$$

Actually, the parameter  $WGF$  can be assigned values of the enumeration type  $\{\text{slow}, \text{medium}, \text{fast}\}$ . The default assignments to the values  $\text{slow}$ ,  $\text{medium}$ , and  $\text{fast}$  are 1.2, 1.3, and 1.4.

## 5 Experimental Results

We performed the following three types of experiments:

1. We made an analysis in which we determined which of the edges that are fixed during our approach are actually present in an optimal solution.
2. We investigated the reduction rates reached by our approach.
3. We applied our approach to some huge TSP instances and compared costs (and running times, if possible) to Helsgaun's LKH.

<sup>8</sup>  $D$  is an abbreviation for **DISPLACEMENT**

<sup>9</sup>  $MNL$  is an abbreviation for **MIN\_NODE\_LIMIT**

<sup>10</sup>  $WGF$  is an abbreviation for **WINDOW\_GROWTH\_FACTOR**

**Table 1.** How many edges which are fixed are contained in a given optimal solution? The second column shows the number of cities of the instances, the third column the number of edges fixed by our approach, and the fourth one the number of fixed edges which are contained in a given optimal solution. The displacement  $D$  had been set to  $1/2$  during this experiment. The experiment has been performed on a standard personal computer.

instance	size	fixed edges	thereof optimal	fraction	runtime
Greece	9 882	4,739	4,436	93,61 %	475 s
Finland	10,639	4,840	4,655	96,18 %	376 s
Italy	16,862	10,180	9,714	95,42 %	876 s
Vietnam	22,775	11,445	10,820	94,54 %	1,602 s
Sweden	24,978	16,523	15,470	93,63 %	1,502 s

Our first experiment deals with the question of how many edges are fixed by our approach and how many of these fixed edges are contained in at least one optimal tour. Unfortunately, we couldn't perform this experiment on huge TSP instances, but had to use middle-sized TSP instances consisting of "only" some thousands of vertices as we need to know the optimal solutions of the instances for being able to compute the number of fixed edges contained in optimal tours. Actually, as we also do not know *all* optimal solutions of middle-sized TSP instances, we counted the number of fixed edges contained in *one given* optimal solution, i. e., we computed a lower bound of the number of fixed edges contained in optimal solutions. Table 1 shows the result of this experiment with respect to the national TSP instances Greece (9,882 cities), Finland (10,639 cities), Italy (16,862 cities), Vietnam (22,775 cities), and Sweden (24,978 cities). The table gives us an impression of how many of the edges which are fixed by our approach are contained in an optimal tour, namely between 93 % and 96 %. Remember, this is only a lower bound; the ratio could be yet much higher. In the instances of Table 1 the ratio between the number of fixed edges and the size of the instance which is between 48 % and 66 % (see the second and third column of Table 1) is unsatisfactory. Note that this unsatisfactory percentage is due to the fact that the TSP instances taken in this experiment are relative small and we set the parameter value  $MNL$  to about 1,000 which is a reasonable value for this parameter when the approach is applied to such middle-sized TSP instances. However, this yields comparatively many trivial windows so that comparatively many edges are excluded from becoming a pseudo backbone. Table 2 exemplarily shows how many cities are typically eliminated during each of the iterations when the approach is iteratively applied to huge TSP instances. (Note that the number of eliminated cities is a lower bound for the number of fixed edges.) For the `World-TSP` which consists of 1,904,711 cities we obtain a size reduction of 98 % from 1,904,711 vertices to 33,687 vertices.

Currently, the best known tour for `World-TSP` has been found by Keld Helsgaun using LKH. The length of this tour is 7,515,877,991 which is at most 0,0487% greater than the length of an optimal tour, as the currently best lower

**Table 2.** How many edges are fixed by our approach when applied to a huge instance? Information about the reduction rates reached by our approach when applied to the `World-TSP`. After 14 iterations, LKH is directly applied to the remaining TSP instance consisting of 33,687 cities.

Iteration	Number of cities eliminated in this iteration	Number of paths contracted in this iteration	Size of the new TSP instance
1	103,762	20,811	1,800,949
2	228,887	56,007	1,572,062
3	325,034	87,511	1,247,028
4	327,625	100,443	919,403
5	271,871	95,234	647,532
6	194,280	80,672	453,252
7	148,191	62,884	305,061
8	94,678	47,089	210,383
9	70,599	34,432	139,784
10	43,061	24,673	96,723
11	21,082	19,292	75,641
12	18,830	14,574	56,811
13	15,265	11,286	41,546
14	7,859	9,496	33,687

bound for the tour length of the `World-TSP` is 7,512,218,268 (see [19]). However, no overall running time comprising the running time of both, the computation of good starting tours *and* the iterative  $k$ -opt steps of LKH are reported. Helsgaun reports that by assigning the right values to the parameters, LKH computes a tour for the `World-TSP` in an hour or two<sup>11</sup> which is at most 0,6% greater than the length of an optimal tour [7]. By using sophisticated parameters, LKH even finds better tours (up to only 0.12% greater than the length of an optimal tour) in a couple of days [9].

We applied our iterative approach to the `World-TSP` instance, too. Using appropriate parameter values our approach constructs a tour for `World-TSP` of length 7,525,520,531 which is at most 0,18% above the lower bound in less than two days on a standard personal computer and a tour of length 7,530,566,694 which is at most 0,2442% greater than the length of an optimal tour in about 7 hours; it computes a tour with length 7,524,796,079 (gap=0.1674%) in less than 13 hours on a parallel computer with 32 Intel Xeon 2.4 GHz processors. Actually, we exploited one of the central properties of our iterative approach, namely the property that the approach can be highly parallelized, as the tours for the windows of an iteration can be computed in parallel.

Table 3 shows our experimental results with respect to `World-TSP` for both, runs on a standard personal computer and runs on the above mentioned parallel machine. Table 4 summarizes the results with respect to two instances of the

<sup>11</sup> The computation time of 1,500 seconds, Helsgaun states in [7, pages 92-93], does not include the computation time of the starting tour [9].

**Table 3.** Trade-off between runtime and tour length. The table presents the best experimental results which we obtained by our approach applied to **World-TSP**. Column 6 gives the gaps between the lengths of the tours with respect to the lower bound 7,512,218,268 on the minimum tour length.

<i>standard personal computer</i>						
D	MNL	IWS	WGF	tour length	gap [%]	runtime
1/3	20,000	40	slow	7,520,207,210	0,1063 %	6 days 21 hours
1/3	10,000	30	slow	7,521,096,881	0,1181 %	5 days 12 hours
1/3	10,000	17	slow	7,522,418,605	0,1357 %	5 days 8 hours
1/2	20,000	18	medium	7,525,520,531	0,1770 %	1 day 21 hours
1/2	20,000	18	medium	7,528,686,717	0,2192 %	1 day 15 hours
1/2	5,000	40	medium	7,529,946,223	0,2359 %	1 day 10 hours
1/2	5,000	100	medium	7,533,272,830	0,2802 %	1 day 10 hours

<i>parallel computer with 32 Intel Xeon 2.4 Ghz processors</i>						
D	MNL	IWS	WGF	tour length	gap [%]	runtime
1/2	20,000	50	medium	7,524,796,079	0,1674 %	0 day 13 hours
1/2	20,000	15	medium	7,529,172,390	0,2256 %	0 day 11 hours
1/2	10,000	30	fast	7,530,566,694	0,2442 %	0 day 7 hours

**Table 4.** Best experimental results of our approach applied to the **DIMACS** instances E3M.0 and E10M.0 [15]. Column 8 gives the gap to the current best known tour [15].

<i>standard personal computer</i>								
	size	D	MNL	IWS	WGF	tour length	gap [%]	runtime
E3M.0	3,162,278	1/3	10,000	16	slow	1,267,959,544	0,0465 %	5.71 days
		1/3	10,000	20	slow	1,268,034,733	0,0525 %	5.46 days
		1/2	20,000	12	slow	1,268,280,730	0,0719 %	4.08 days
		1/2	10,000	30	medium	1,268,833,812	0,1156 %	1.62 days
E10M.0	10,000,000	1/3	10,000	30	slow	2,254,395,868	0,0541 %	19.67 days
		1/2	10,000	30	medium	2,256,164,398	0,1326 %	4.25 days
		1/2	5,000	40	slow	2,260,519,918	0,3259 %	2.62 days

DIMACS TSP Challenge which have sizes of 3,162,278 and 1,000,000 vertices, respectively [15]. Note the general high-level trade-off provided by the approach:

- the smaller the factor D by which the window frame is shifted, the better the tours and the worse the runtime, although the runtimes remain acceptable;
- the faster the increase of the window frame, the better the runtime and the worse the tours, although the tour lengths remain very good.

## 6 Future Work

One problem with the approach presented in this paper arises, if the vertices are non-uniformly distributed, i. e., if there are both, regions with very high densities

and regions with very low densities. In order to partition the regions of high density in such a way that LKH can handle the windows efficiently with respect to both, tour quality and running time, the parameter IWS should be large. However this results in many trivial windows located in regions of low density. To overcome this problem, we reason about recursion which can be applied to windows containing too much vertices and about non-uniform clustering of dense regions.

## References

1. D.L Applegate, R.E. Bixby, V. Chvátal, W.J. Cook: The Traveling Salesman Problem. A Computational Study. *Princeton University Press*, 2006.
2. D.L Applegate, R.E. Bixby, V. Chvátal, W.J. Cook, D. Espinoza, M. Goycoolea, K. Helsgaun: *Certification of an Optimal Tour through 85,900 Cities*. *Operations Research Letters* 37(1):11-15, 2009.
3. W. Cook, P. Seymour: *Tour Merging via Branch-Decomposition*. *INFORMS Journal on Computing* 15(3):233-248, 2003.
4. C. Dong, G. Jäger, D. Richter, P. Molitor: *Effective Tour Searching for TSP by Contraction of Pseudo Backbone Edges*. In: Proc. of AAIM'2009, 175-187, San Francisco, 2009.
5. G. Gutin, A.P. Punnen (Eds.): *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, Dordrecht, 2002.
6. K. Helsgaun: *An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic*. *European Journal Operations Research* 126(1):106-130, 2000.
7. K. Helsgaun: *An Effective Implementation of k-opt Moves for the Lin-Kernighan TSP Heuristic*. *Writings in Computer Science* 109, Roskilde University, 2006.
8. K. Helsgaun: *General k-opt submoves for the Lin-Kernighan TSP heuristic*. *Mathematical Programming Computation* 1(2-3):119-163, Springer 2009.
9. K. Helsgaun. Private Communication, March 2009.
10. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (Eds.): *The Traveling Salesman Problem - A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, Chicester, 1985.
11. S. Lin, B.W. Kernighan: *An Effective Heuristic Algorithm for the Traveling Salesman Problem*. *Operations Research* 21:498-516, 1973.
12. C. Walshaw: *A Multilevel Lin-Kernighan-Helsgaun Algorithm for the Travelling Salesman Problem*. TR 01/IM/80, Computing and Mathematical Sciences, University of Greenwich, UK, 2001.
13. W. Zhang, M. Looks: *A Novel Local Search Algorithm for the Traveling Salesman Problem that Exploits Backbones*. In: Proc. of the 19th Int'l Joint Conf. on Artificial Intelligence (IJCAI-05), 343-350, 2005.
14. TSPLIB Homepage: <http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html>.
15. DIMACS Implementation Challenge: <http://www.research.att.com/~dsj/chtsp/>.
16. National Instances: <http://www.tsp.gatech.edu/world/summary.html>.
17. TSP Homepage: <http://www.tsp.gatech.edu/>.
18. VLSI Instances from the TSP Homepage: [www.tsp.gatech.edu/vlsi/summary.html](http://www.tsp.gatech.edu/vlsi/summary.html).
19. World-TSP from the TSP Homepage: [www.tsp.gatech.edu/world/](http://www.tsp.gatech.edu/world/).
20. Source Code of [1] (Concorde): <http://www.tsp.gatech.edu/concorde/index.html>.
21. Source Code of [7] (LKH): <http://www.akira.ruc.dk/~keld/research/LKH/>.