Finding Good Tours for Huge Euclidean TSP Instances by Iterative Backbone Contraction: First Results

Christian Ernst, Changxing Dong, Dirk Richter, Gerold Jäger, Paul Molitor
Finding Good Tours for Huge Euclidean TSP Instances
by Iterative Backbone Contraction: First Results
Christian Ernst, Changxing Dong, Dirk Richter, Gerold Jäger, Paul Molitor
This paper presents an iterative approach to find good tours for very large instances of the well-known Euclidean 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, 9, 16] 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 shortest Hamiltonian cycle of length n. In this paper we consider Euclidean TSP (ETSP) instances whose cities are embedded in the Euclidean plane\(^1\).

Although TSP is easy to understand, it is hard to solve, namely \(NP\)-hard. We distinguish two classes of algorithms for the symmetric TSP, namely heuristics and exact algorithms. For the exact algorithms the program package Concorde [1, 19], which combines techniques of linear programming and branch-and-cut, is the currently leading code. Concorde has exactly solved

\(^1\)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 [18]).
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, 7, 20] (LKH), which is an effective implementation of the Lin-Kernighan heuristic [10], is one of the best packages. Especially for the most yet not exactly solved TSP benchmark instances [13, 14, 15, 17, 18] this code found the currently best tours.

An interesting observation [12] 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 [17] with sizes up to 47,608, this approach found better tours than the best ones so far reported in literature.

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 [11]. 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. Furthermore Walshaw’s approach seems not to be applicable to very large problem instances.

An alternative to this 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 [10] and is implemented in LKH. The main difference to the approaches presented in [4, 11] 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 heuristics for TSP during the computation of the starting tours 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 incremental shift is chosen as decimal 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 2 where the basic idea is illustrated for $s = 2$). 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, 11], 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 are 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, the approach computes a tour for World-TSP of length 7,524,796,079, which is only 0.16% above a known lower bound, within 13 hours on a standard personal computer. Similarly, it finds a tour for the DIMACS 3,162,278-sized TSP instance [14], 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 $s$ which determines the shift amount of the window frame, 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 algorithms are listed in Section 4. Section 5 presents the experimental results. Finally, conclusions and suggestions for future work are given in Section 6.
2 Definitions

2.1 Basics

Let $V$ be a set of $n$ vertices embedded in the Euclidean plane (see Figure 1) and let $\text{dist}(u, w)$ be the Euclidean distance between the vertices $u$ and $w$. A sequence $p = (v_1, v_2, \ldots, v_q)$ with \(\{v_1, v_2, \ldots, v_q\} \subseteq V\) is called path of length $q$. The costs $\text{dist}(p)$ of such a path $p$ is given by the sum of the Euclidean distances between neighboring vertices, i.e., $\text{dist}(p) := \sum_{i=1}^{q-1} \text{dist}(v_i, v_{i+1})$.

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 $p = (v_1, v_2, \ldots, v_n)$ can be extended to the closed path $T = (v_1, v_2, \ldots, v_n, v_1)$. Such a closed path of $V$ is called tour.
2.2 Euclidean Traveling Salesman Problem

ETSP (Euclidean Traveling Salesman Problem) 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 \( FE \subseteq \{\{v_i, v_j\} : v_i, v_j \in V \) and \( v_i \neq v_j\} \) of fixed edges, if a tour \( T = (v_1, v_2, \ldots, v_n, v_1) \) has to be computed such that

1. \( (\forall\{u, w\} \in FE)(\exists i \in \{1, \ldots, n\}) \{v_i, v_{(i+1) \mod n}\} = \{u, w\} \)
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 \( p = (v_i, v_{i+1}, \ldots, v_{j-1}, v_j) \) transforms a constrained ETSP instance \((V, FE)\) into a constrained ETSP instance \((V', FE')\) with \( V' = V \setminus \{v_{i+1}, \ldots, v_{j-1}\} \) and \( FE' = (FE \cup \{\{v_i, v_j\}\}) \cap (V' \times V') \). Thus the inner vertices \( v_{i+1}, \ldots, v_{j-1} \) of path \( p \) are deleted —only the boundary vertices \( v_i, v_j \) of \( p \) remain in the new constrained ETSP instance— and the size of the new constrained ETSP becomes smaller, unless \( i+1 = j \). Figures 5(b) and 5(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 you can see in Figure 3, 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 \textsc{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, 20] or by the exact solver Concorde [19]. Since LKH finds optimal solutions frequently for small instances and the sizes of the windows are rather small, we do not apply Concorde.
Figure 3: Moving a window frame across the bounding box with $\text{WINDOW.SCALE}=2$ and $s=2$. Each window defines a sub-problem for which a good tour is computed, independently of the neighboring sub-problems.

The sub-problems should contain a number of vertices greater than a lower bound $MNL$ 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 \textit{trivial} instance. A tour is computed for each of the non-trivial sub-problems. Figure 4 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\footnote{If one of the windows containing $u$ and $w$ is trivial, the edge $\{u, w\}$ is not considered as pseudo-backbone.} —for $s=2$, in some sense, the four windows together reflect the surrounding area of the common vertices with respect to the four directions. We call such an edge $\{u, w\}$ \textit{pseudo backbone edge}.\footnote{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.}

Figures 4 and 5(a) illustrate the notion of pseudo backbone edges. For this purpose, consider
Figure 4: A good tour is computed for each of the non-trivial sub-problems. We have set parameter $MNL$ to 3, so that there are no trivial instances.

only the four top left-hand windows. Figure 4 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 (see Figure 4) is shown in Figure 5(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 5(c)).

This process is iteratively repeated. The parameter $\text{WINDOW\_SCALE}$ which determines the width and height of the window frame is re-adjusted, i.e., decreased, in each iteration —see Section 4 for more details. The iteration stops if $\text{WINDOW\_SCALE}$ 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 $\text{IWS}^4$ which determines the width and height of the window frame applied in the first iteration, namely

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

with $\text{WINDOW}_\text{SCALE} := \text{IWS}$.

2. $\text{D}^5$ which is the shift amount relative to the width (height) of the window frame by which the window frame is shifted, i.e., $\text{D} = 1/s$.

3. The minimum number $\text{MNL}^6$ 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 $\text{MNL}$ no tour of the vertices contained in that window is computed, as those tours would be suitable to only a limited extent for finding good pseudo backbone edges.

4. The parameter $\text{WGF}^7$ which is the factor by which the window frame is re-adjusted after each iteration, i.e.,

$$\text{WINDOW}_\text{SCALE}_{\text{new}} = \frac{\text{WINDOW}_\text{SCALE}_{\text{old}}}{\text{WGF}}.$$ 

Actually, the parameter $\text{WGF}$ can be assigned values of the enumeration type $\{\text{slow}, \text{normal}, \text{fast}\}$.

The default assignments to the values $\text{slow}$, $\text{normal}$, and $\text{fast}$ are 1.2, 1.3, and 1.4.

---

$^4$IWS is an abbreviation of INITIAL WINDOW SCALE

$^5$D is an abbreviation of DISPLACEMENT

$^6$MNL is an abbreviation of MIN NODE LIMIT

$^7$WGF is an abbreviation of WINDOW GROWTH FACTOR
5 Experimental Results

We performed the following three types of experiments:

1. First of all, 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. This experiment could be performed only on middle-sized TSP benchmarks as we had to know optimal tours of the TSP instances used as benchmarks to conduct this experiment.

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.

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 TSPs of Greece (9,882 cities), Finland (10,639 cities), Italy (16,862 cities), Vietnam (22,775 cities), and Sweden (24,978 cities). The third column of 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.

You might be dissatisfied with 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). Note that this unsatisfactory percentage is due to the fact that the TSP instances taken in this experiment are relative small and we set parameter value \( \text{MNL} \) to about 1,000 which is a reasonable value for this parameter when the approach is applied to such middle-sized TSP-instances. 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
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.

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

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.

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

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 bound for the tour length of the World-TSP is 7,512,218,268 (see [18]). 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\textsuperscript{8} which is at most 0.6\% greater than the length of an optimal tour. By using better parameters, LKH even finds a tour at most 0.25\% longer than the length of an optimal tour in two days \cite{8}.

We applied our iterative approach to the \texttt{World-TSP} instance, too. Using appropriate parameter values our approach constructs a tour for \texttt{World-TSP} of length 7,530,566,694 which is at most 0.2442\% greater than the length of an optimal tour in about 7 hours and 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 \texttt{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 DIMACS TSP Challenge which have sizes of 3,162,278 and 1,000,000 vertices, respectively \cite{14}. Note the high-level trade-off provided by the approach:

- the smaller the amount $D$ by which the window frame is shifted, the better are the tours and the worse is the runtime, although the runtimes remain acceptable;

- the faster the increase of the window frame, the better is the runtime and the worse are the tours, although the tour lengths remain very good.

\textsuperscript{8}The computation time of 1,500 seconds, Helsgaun states in \cite[pages 92-93]{7}, does not include the computation time of the starting tour \cite{8}.
Table 3: Trade-off between runtime and tour length. The table presents 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.

<table>
<thead>
<tr>
<th>D</th>
<th>MNL</th>
<th>IWS</th>
<th>WGF</th>
<th>tour length</th>
<th>gap [%]</th>
<th>runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3</td>
<td>20,000</td>
<td>40</td>
<td>slow</td>
<td>7,520,207,210</td>
<td>0.1063 %</td>
<td>6 days 21 hours</td>
</tr>
<tr>
<td>1/3</td>
<td>10,000</td>
<td>30</td>
<td>slow</td>
<td>7,521,096,881</td>
<td>0.1181 %</td>
<td>5 days 12 hours</td>
</tr>
<tr>
<td>1/3</td>
<td>10,000</td>
<td>17</td>
<td>slow</td>
<td>7,522,418,605</td>
<td>0.1357 %</td>
<td>5 days 8 hours</td>
</tr>
<tr>
<td>1/2</td>
<td>20,000</td>
<td>18</td>
<td>medium</td>
<td>7,525,520,531</td>
<td>0.1770 %</td>
<td>1 day 21 hours</td>
</tr>
<tr>
<td>1/2</td>
<td>20,000</td>
<td>18</td>
<td>medium</td>
<td>7,528,686,717</td>
<td>0.2192 %</td>
<td>1 day 15 hours</td>
</tr>
<tr>
<td>1/2</td>
<td>5,000</td>
<td>40</td>
<td>medium</td>
<td>7,529,946,223</td>
<td>0.2359 %</td>
<td>1 day 10 hours</td>
</tr>
<tr>
<td>1/2</td>
<td>5,000</td>
<td>100</td>
<td>medium</td>
<td>7,533,272,830</td>
<td>0.2807 %</td>
<td>1 day 10 hours</td>
</tr>
</tbody>
</table>

Table 4: Best experimental results which we obtained by our approach applied to DIMACS TSPs [14]. Here, column 8 gives the gap to the current best known tour [14].

<table>
<thead>
<tr>
<th>instance</th>
<th>size</th>
<th>D</th>
<th>MNL</th>
<th>IWS</th>
<th>WGF</th>
<th>tour length</th>
<th>gap [%]</th>
<th>runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>E3M.0</td>
<td>3,162,278</td>
<td>1/3</td>
<td>10,000</td>
<td>16</td>
<td>slow</td>
<td>1,267,959,544</td>
<td>0.0465 %</td>
<td>5 days 17 hours</td>
</tr>
<tr>
<td>E10M.0</td>
<td>10,000,000</td>
<td>1/3</td>
<td>10,000</td>
<td>30</td>
<td>slow</td>
<td>2,254,395,868</td>
<td>0.0541 %</td>
<td>19 days 16 hours</td>
</tr>
</tbody>
</table>

parallel computer with 32 Intel Xeon 2.4 Ghz processors

<table>
<thead>
<tr>
<th>D</th>
<th>MNL</th>
<th>IWS</th>
<th>WGF</th>
<th>tour length</th>
<th>gap [%]</th>
<th>runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2</td>
<td>20,000</td>
<td>50</td>
<td>medium</td>
<td>7,524,796,079</td>
<td>0.1674 %</td>
<td>0 day 13 hours</td>
</tr>
<tr>
<td>1/2</td>
<td>20,000</td>
<td>15</td>
<td>medium</td>
<td>7,529,172,390</td>
<td>0.2256 %</td>
<td>0 day 11 hours</td>
</tr>
<tr>
<td>1/2</td>
<td>10,000</td>
<td>30</td>
<td>fast</td>
<td>7,530,566,694</td>
<td>0.2442 %</td>
<td>0 day 7 hours</td>
</tr>
</tbody>
</table>
6 Conclusion and Future Work

In this paper we have proposed a new approach of computing good tours of huge TSP problems in reasonable time. The experiments presented prove its efficiency.

One problem with the approach 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 a recursive approach which can be applied to windows containing more than $\text{MAX\_NODE\_LIMIT}$ vertices which would be a further parameter of the algorithm.

References

[16] TSP Homepage:
   http://www.tsp.gatech.edu/.

[17] VLSI Instances from the TSP Homepage:
   www.tsp.gatech.edu/vlsi/summary.html.

[18] World-TSP from the TSP Homepage:
   www.tsp.gatech.edu/world/.


   http://www.akira.ruc.dk/~keld/research/LKH/.