Solving for large-scale traveling salesman problem with divide-and-conquer strategy

Kazunori Ishikawa, Ikuo Suzuki, Masahito Yamamoto, Masashi Furukawa
Graduate School of Information Science and Technology, Hokkaido University
Kita 8-jo Nishi 5-chome, Sapporo, Hokkaido, Japan
Email:k-ishikawa@complex.eng.hokudai.ac.jp

Abstract—We propose a new divide-and-conquer method for solving large-scale Traveling Salesman Problem. The main feature of the method is to construct some child-problems and child-tours combined with some local parts of a tour. A solution is optimized widely by repeating optimization of each child-tours. Child-tours can be optimized individually, so our method may be applied parallel computing. In experiments, the method Local Clustering Organization is used to optimize child-tours. Reasonably approximated solutions are obtained in useful computational times. And our method shows superiority to large-scale TSP in convergence time of solution.

I. GENERAL INTRODUCTIONS

Traveling Salesman Problem (TSP) is one of the most studied problems in the field of computer science. TSP is a problem to find the shortest Hamiltonian cycle, where a list of cities and distances between them are given. This problem has many engineering applications, for example, finding efficient wire-bonding and boring path, and designing VLSI layout. The scale of applications has increased in recent years. However, TSP has been regarded as a NP-hard problem, which seems to be impossible to solve in polynomial times by deterministic Turing machines. In fact, there is no algorithm which obtains exactly optimal solution in $O(n)$ time. So it is almost impossible to find an optimal solution for large-scale TSP in reasonable computational time, and there are many studies for approximate solver to obtain reasonably approximated solution. Our study also aims to obtain an approximate solution for large-scale TSP in practically enough computation time.

II. TRAVELING SALESMAN PROBLEM

TSP is one of the most popular combinatorial optimization problem. It gives us some cities and distances between all of them, and requires to find a tour which has the least cost.

The size of a problem is represented in the number of cities. If a problem has $n$ cities, then the size of the problem is $n$. A tour of a TSP is one of the solutions of the problem, and it means a Hamiltonian cycle, a round way that visits each city exactly once. A path between two cities in a tour is called edge. There are $(n-1)!/2$ patterns of a tour if the problem has $n$ cities. In TSP, all local costs between two cities are previously defined. The total cost of a tour $c_t$ is defined as a sum of all local costs of the edges in the tour. Calculation of $c_t$ is represented by Eq.(1), where $N$ is a set of cities, $n$ is the number of cities, $c_{st}$ is a local cost between city $s$ and $t$, and $o(p)$ is the $p$-th city in the tour.

$$c_t = \sum_{i=0}^{n-2} c_{o(i) o(i+1)} + c_{o(n-1) o(0)} \quad (o \in N) \quad (1)$$

In Eq.(1), $o(0)$ is the start city, and $o(n-1)$ is the final city in the tour. $c_{o(n-1) o(0)}$ is the local cost to back to the first city from the final city in the tour.

A general type of TSP defines local costs in $n \times n$ matrix. And there are some variations in TSP. The metric TSP is what all local cost satisfies a triangle inequality. The following is a mathematical expression of triangle inequality (2).

$$c_{ij} \leq c_{jk} + c_{kj} \quad (\forall i, \forall j, \forall k \in N) \quad (2)$$

Especially, a variation of TSP called the Euclidean TSP give us cities with their x and y-coordinate, and Euclidean distances are calculated as the costs between each cities. This variation of TSP has been widely studied. In this paper we treat only Euclidean TSP because it is hard to set aside cost matrix for large-scale TSP. In fact, there are no benchmarks on a general type of TSP which has over 10000 cities. Euclidean TSP is also NP-hard problem if coordinate informations are not used except the calculations of costs.

III. RELATED STUDIES

A. Heuristics

There are studies for solving large-scale TSP with advanced application of existing approximate solvers such as Genetic Algorithm, Simulated Annealing, Tabu Search, Self Organizing Map and Lin-Kernighan algorithm. These solvers were originally invented for solving middle or small-scale TSP, and in the following studies, they are modified, tuned up or parallelized for large-scale TSP [1], [2], [3].

B. Divide-and-Conquer

Divide-and-Conquer (D&C) is a strategy to solve a large-scale problem by dividing it into some small problems then solving each of them. In a study for solving large-scale TSP with D&C strategy, the experiments shows that this strategy allows us to solve TSP quickly and to increase scalability. Nevertheless, most of these methods need to use informations on coordinates of cities for divide a problem and constructing small child-problems. So they can be applied only to Euclidean TSP. It is impossible for them to apply to general ones, in which costs between cities are defined in a matrix [4], [5].
IV. LOCAL CLUSTERING ORGANIZATION

Local Clustering Organization (LCO) is an approximate method originally investigated to solve TSP in short [6]. In this method, local optimization procedures are repeated to achieve a globally optimized state of solution. LCO was designed to use the same idea as the synapse learning rule of SOM, but LCO can be also applied to the general TSP unlike SOM. In the LCO-algorithm applied TSP, neurons are regarded as each cities and linked annularly. Local combinational neuron orders are repeatedly optimized by arbitrarily defined clustering methods.

LCO has been applied not only TSP but also the other combinatorial optimization problem, such as n-TSP, Vehicle Routing Problem, Jobshop Scheduling Problem, and Quadratic Assignment Problem [7].

A. Clustering method

LCO uses some parameters as followings. \( c \) is a base neuron to which clustering methods are applied. \( r \) is the range to which a clustering method effects. \( N_c \) is a set of neighborhood neurons of \( c \). \( N_c \) contains all neurons which are in front and back \( r \) from the neuron \( c \) on the annular topology.

A clustering method is randomly chosen in arbitrary rate, and applied to \( c \) and all neurons in \( N_c \). The followings are the clustering methods generally used to solve TSP.

1) Simple Exchange Method, SEM: This method repeats exchanging orders of two neurons. At first, two neurons are selected, \( c \) and its next neuron \( c+1 \). Then the locations of the two neurons are swapped if the cost down after this swapping. Second, \( c \) and \( c-1 \), the neuron before \( c \), are selected. And the former swap operation is applied to the selected neurons. Afterward, the selections continues int the order \((c,c+2),(c,c-2),(c,c+3),(c,c-3)\) and the final selection is \((c,c-r)\). The swap rule is applied to a pair of neurons repeatedly.

2) Inverse Exchange Method, IEM: A partial order of neurons is inversed repeatedly by this method. Neuron selections are the same as the method SEM. In every step of this method, the order of neurons from \( c \) to \( c+i \) is reversed if the cost down. In other words, this method applies “2-opt” to the selected neurons.

3) Smoothing Method, SM: This method runs 2-opt to all applicative patterns in \( N_c \). The selection of a pair of neurons starts from the beginning of \( N_c \). A pair \((c-r,c-r+1)\) is selected and applied 2-opt at first. Then pairs \((c-r,c-r+i)\) \((2 \leq i \leq 2r)\) are selected until \((c-r,c+r)\) are selected. After applying 2-opt for \((c-r,c+r)\), the pair \((c-r+1,c-r+2)\) is selected and selections continue in turn. The all patterns of selections are \((c-r+i,c-r+j)\) \((0 \leq i \leq 2r, i < j \leq 2r)\). SM runs 2-opt \((2r^2+r)\) times and this method requires more computational cost than SEM and IEM do. So the rate employed for SM is generally smaller than one for SEM and IEM.

B. Algorithm applied to TSP

A general algorithm of LCO applied to TSP is illustrated under-below.

1) Connect neurons by the same number of cities annularly.

2) Update the value of clustering-radius \( r \).

3) Select a clustering-base neuron \( c \) randomly.

4) Apply a clustering method to \( c \) and its neighbor \( N_c \).

5) Stop the algorithm if a given termination condition is satisfied, otherwise returns to 2).

One turn of the algorithm is defined as the procedures from 2) to 5).

In general, the value of \( r \) is initially set to be small like 1. Then \( r \) is increased up to the half of the number of cities, \( n/2 \). For example, we can set \( r \) initially 1, and in every turn it is increased by 1%.

V. A PROPOSED DIVIDE-AND-CONQUER METHOD

Based on the strategy of D&C, the proposal method generates some small problems from a current tour. At this time, coordinate informations of cities are not used at all so that the method can be applied to the general type of TSP.

In this D&C method, the term “problem” means a set of cities given in TSP. The “tour” is a solution for TSP. The “sub-problem” is equivalent to a sub-set of problem which is obtained by dividing a problem. The “sub-tour” is a part of a tour obtained by removing some edges in the tour. And the “child-problem” is a set of sub-problems, and we call its solution “child-tour”. A child-problem can be treated as an usual TSP without two exceptions described in the later section. Optimizing reconstructed child-tour repeatedly leads to a widely optimized tour. This D&C assume that methods to solve child-problems use local search technique. Fig.1 shows the overview of the method.

In the following sections, it is described that the detail procedure of constructing, optimizing, and merging child-problems and child-tours.

A. Constructing child-tours

A child-problem consists of some sub-problems of an original problem. It has an initial child-tour constructed by combing some sub-tours. A sub-tour is a chain of edges, which is a consecutive part of an original tour.

To construct initial child-tour, sub-problems and sub-tours are generated by dividing the original tour at first. A tour is divided into sub-tours which contains the same number of
Fig. 2a. An example of constructing initial child-tours

Fig. 2b. An example of optimizing child-tours

Fig. 2c. An example of merging child-tours

cities by removing edges from a randomly selected edge. It is not clear whether dividing a tour equally is the most proper way to constructing an initial child-tour, and this is the rest for a future research.

Then, after generating sub-tours, sub-tours are concatenated by defining virtual edges so that a child-tour can be regarded as a tour of TSP. Each sub-tours are linked at their both ends by virtual edges. Sub-tours to be combined are also chosen randomly. It is impossible to optimize a tour widely by optimizing only each sub-tours because the max range of optimization is at most the size of a sub-problem. To conquer this issue, some sub-tours are combined and optimized at the same time in a child-tour. This is why combining sub-tours in a child-tour.

There exists the relation among the number of child-problem $s$, the number of combining sub-problems $u$, and the number of edge $d$ to divide an original tour.

$$d = s \cdot u$$

(3)

Fig.2a shows an example of constructing initial child-tours under a parameter $s = 2$, $u = 3$, $d = 6$.

B. Optimizing child-tours

Starting from initial child-tours, they are optimized by arbitrary solvers. A child-tour can be regarded as an usual tour of TSP without two exceptions below.

First, virtual edges which are defined in constructing child-tour is fixed at its position and not allowd to remove (Fig.2b).

This is because they don’t exist in the original tour before dividing, so they should not be dealt in optimization. In addition, operations to merge child-tours after optimizing them become complex if locations of virtual edges are changed. Keeping virtual edges’ location helps to merge child-tours easily.

The second exception arises if a child-tour is optimized with certain operations which depend on the order of sub-tours’ sequence. It is necessity to manage and check informations in sub-tours’ sequence. For example, the operation called “2-opt” removes and redefines two edges if a cost decrease after operation. But as shown an example in Fig.3, there are two variations in redefining edges and these variations depend on the sub-tours’ sequence.

Without these two exceptions, a child-tour can be optimized as an usual tour of TSP.

C. Merging child-tours

Optimized child-tours are merged into a tour and then an new optimized tour is obtained. To merge child-tours, they are divided by removing virtual edges at first and this dividing operation generates $d$ optimized sub-tours. After that, merge all sub-tours by putting back edges which are removed when generating sub-tours before the construction of initial child-tours. A complete tour as an new solution is obtained by this merging operation. An example of merging child-tours is shown in Fig.2c.

D. Algorithm

The general procedure to optimize TSP with a proposed D&C method is as follows. It is supposed that the parameters $s$, $u$ and $d$ are given previously.

1) Create a tour as an initial solution by an proper method.
2) Generate $d$ sub-tours from the current tour.
3) Generate $s$ child-tours by uniting $u$ sub-tours randomly.
4) Apply arbitrary TSP solvers to child-tours.
5) Merge all child-tours into a tour as an new solution.
6) Stop if termination conditions are satisfied, otherwise returns to 2).

In this method, one turn means a set of procedures from 2) to 6).

VI. NUMERICAL EXPERIMENTS TO ANALYZE AND TUNE PARAMETERS

Two parameters, the number of child-tours $s$ and the number of combining sub-tours $u$, are expected to have a great effect with the proposed D&C method. In this section, we experiment to examine how the proposed method is affected by these parameters, solving large-scale TSPs. Two benchmark problems
TABLE I
PARAMETERS FOR LCO TO OPTIMIZE CHILD-TOURS

<table>
<thead>
<tr>
<th>clustering method (selection rate)</th>
<th>SEM, IEM, SM,</th>
<th>(40%)</th>
<th>(40%)</th>
<th>(20%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_i )</td>
<td>min { 1, 0.7, ( n_s/2 ) }</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>termination conditions</td>
<td>running ( n_s^{1/2} ) steps</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE II
RELATION BETWEEN \( n, s, \) AND \( n_s \)

\[
\begin{align*}
\text{when } \ n &= 85900 \text{ (pla85900) } & s &= n^{1/2} & s &= n^{1/3} & s &= n^{1/4} \\
& & s &= 293 & s &= 44 & s &= 17 \\
& & n_s &= 293 & n_s &= 1952 & n_s &= 5052 \\
\text{when } \ n &= 744710 \text{ (lrb744710) } & s &= n^{1/2} & s &= n^{1/3} & s &= n^{1/4} \\
& & s &= 862 & s &= 90 & s &= 29 \\
& & n_s &= 863 & n_s &= 8274 & n_s &= 25679
\end{align*}
\]

are used for experiments. Results give us guidelines to make use of the proposed method efficiently.

A. Experimental conditions

Two TSPs, “pla85900” and “lrb744710”, to which are referred on the websites “TSPLIB1” and “The Traveling Salesman Problem2”, are adopted as benchmark problems. This is because these problems have the largest number of cities in these websites, pla85900 has 85900 cities and lrb744710 has 744710 cities. Both problems belong to the Euclidean TSP. All child-tours are optimized by LCO with parameters in Table I, where \( n_s \) means the number of cities in a child-problem, and \( r_i \) is the neighborhood range in the \( i \)-th iteration.

The value of \( s \) is used from \( n^{1/2}, n^{1/3}, \) and \( n^{1/4} \), where \( n \) is the number of cities. Exponential orders of \( n \) are used for \( s \) because large value of \( s \) didn’t provide good results from view points of accuracy and speed, in preliminary experiments. The correspondence relationships \( n, s, \) and \( n_s \) are shown in Table II.

The terminating condition of the proposed D&C is that running time reaches an hour when solving pla85900, and 5 hours when solving lrb744710. The processor used for this experiments is one core of Intel(R) Core(TM)2 Quad CPU Q9550 2.83GHz. The other conditions are described in the experiments is one core of Intel(R) Core(TM)2 Quad CPU Q9550 2.83GHz. The other conditions are described in the experiments.

1) Changing the number of union: At first, the number of child-problems is fixed to \( s = n^{1/2} \). And the number of combine is varied that \( u = 1, 2, 4, 6, 8 \).

2) Changing the number of child-problems: In the next experiment, \( s \) is varied in contrast. \( s \) has values that \( s = n^{1/2}, n^{1/3}, n^{1/4} \), and \( u \) is fixed to \( u = 2 \).

B. Experimental results

1) Changing the number of union: We have the result of the experiments with a set of parameters \( s = n^{1/3}, u = 1, 2, 4, 6, 8 \). Fig.5a shows the relation between the computing time and the cost of solution for pla85900. In the diagram from Fig.5a to Fig.6b, horizontal-axis shows the computing time explained on the second scale. And vertical-axis is costs of solutions. These axes are both logarithmic scale. The cost converges quickly but stays at high value when \( u = 1 \). In case \( u \neq 1 \), the more \( u \) increases, the more cost converges slowly although the converged values are roughly same. Fig.5b is the result in solving the problem lrb744710. It can be said the results have the same characteristic as one of pla85900 while costs with \( u \neq 1 \) are not converged completely in 5 hours.

Fig.4a is a finally obtained tour of pla85900, and Fig.4a is one of lrb744710 with a parameter \( s = n^{1/3}, u = 1 \). Fig.4c is a part of the tour in Fig.4b. Fig.4d is the same part of a tour of lrb744710, when parameter is set to \( s = n^{1/3}, u = 1 \). Note that there are many crossing edges when \( u = 1 \), despite the solution expressing this tour is almost converged.

2) Changing the number of child-problems: Fig.6a is the result of solving pla85900 with the parameter set by \( s = n^{1/2}, n^{1/3}, n^{1/4} \) and \( u = 2 \). There are some periods that each parameters show their superiority. In the earlier time period from the beginning of computation to about 10 seconds, the

---

1. http://comopt.iwr.uni-heidelberg.de/software/TSPLIB95/
2. http://www.tsp.gatech.edu/
Fig. 4d. A part of the lrb744710 tour ($s = n^{1/3}, u = 1$)

Fig. 5a. Cost transition of pla85900 ($s = n^{1/2}$)

Fig. 5b. Cost transition of lrb744710 ($s = n^{1/2}$)

Fig. 6a. Cost transition of pla85900 ($u = 2$)

Fig. 6b. Cost transition of lrb744710 ($u = 2$)

VII. EXPERIMENTS FOR ANALYZE THE FEATURE OF PROPOSAL METHOD

We experiment to examine superiorities of the proposed D&C method. LCO is applied to optimize child-tours generated by the D&C. This hybrid method (D&C hybrid) is compared with LCO in solving large-scale TSPs.

A. Experimental conditions

Benchmark problems to be solved are the same ones in section VI, pla85900 and lrb744710 are adopted as well. The proposed D&C construct $s = n^{1/3}$ child-tours, and $u = 2$ sub-tours are concatenated in a child-tour. This parameter has a good performance in experiments of section VI. Parameters’ values of LCO optimizing child-tours are the same as ones in Table.I. LCO as a comparison method has the same parameters as in Table.I except the terminating condition. The terminating conditions given to D&C hybrid and LCO are that the running time reaches 5 hours.

6 trials are run for each problems.

B. Experimental results

The relations between the cost of solution and computational time are shown in Fig.7a and Fig.7b. horizontal-axis is the computing time and vertical-axis is the cost. All 6 trials are plotted in the same diagrams.

The cost obtained by D&C hybrid decreases faster in early period. The difference of this phenomenon becomes larger when the more larger-scale problem is solved.

Table.III shows the average, best and worst solution accuracies obtained by 6 trials. The value of accuracy indicates what times the obtained cost is larger than the cost of the known best solution in the world. This result shows that proposal method is about 1% worse than LCO in accuracy of solution. This suggest that the D&C strategy accelerate the solving speed but it make the accuracy slightly worse.
The new D&C method which can be applied to general TSPs are proposed. This method generates arbitrary number of child-problems and child-tours as its initial solution, from an arbitrary TSP tour. Global optimization is achieved by optimizing newly generated child-tours repeatedly.

The more the size of problem increase, the more the D&C method shows superiority in solving speed. We can say that this scalability is an advantage of dividing a problem. The proposed method enables to achieve global optimization with relatively-small optimization range, and this feature effects with LCO to optimize faster. This is because the computational costs for the clustering methods in LCO depend on the range of clustering, $r$. The max value of $r$ depends on the size of a problem, $n$. So the total computational cost of LCO depends on $n$. The most computational cost of the other optimization method for TSP depends on $n$, too. It is necessary to investigate whether the D&C method accelerates the solving speed of the other TSP-solver, same as the case of LCO. However, this D&C degrades accuracies of solutions in a few percent. In order to make the accuracy higher, there is necessity to examine the relation between methods employed to optimize the child-tours and the solution accuracy. These works are remained in future.

REFERENCES


The new D&C method which can be applied to general TSPs are proposed. This method generates arbitrary number of child-problems and child-tours as its initial solution, from an arbitrary TSP tour. Global optimization is achieved by optimizing newly generated child-tours repeatedly.

The more the size of problem increase, the more the D&C method shows superiority in solving speed. We can say that this scalability is an advantage of dividing a problem. The proposed method enables to achieve global optimization with relatively-small optimization range, and this feature effects with LCO to optimize faster. This is because the computational costs for the clustering methods in LCO depend on the range of clustering, $r$. The max value of $r$ depends on the size of a problem, $n$. So the total computational cost of LCO depends on $n$. The most computational cost of the other optimization method for TSP depends on $n$, too. It is necessary to investigate whether the D&C method accelerates the solving speed of the other TSP-solver, same as the case of LCO. However, this D&C degrades accuracies of solutions in a few percent. In order to make the accuracy higher, there is necessity to examine the relation between methods employed to optimize the child-tours and the solution accuracy. These works are remained in future.

REFERENCES


TABLE III
ACCURACIES OF OBTAINED SOLUTIONS

<table>
<thead>
<tr>
<th></th>
<th>pla85900</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>aver.</td>
<td>best</td>
<td>worst</td>
<td>aver.</td>
<td>best</td>
<td>worst</td>
</tr>
<tr>
<td>D&amp;C hybrid</td>
<td>1.120</td>
<td>1.115</td>
<td>1.124</td>
<td>1.129</td>
<td>1.213</td>
<td>1.235</td>
</tr>
<tr>
<td>LCO</td>
<td>1.109</td>
<td>1.108</td>
<td>1.111</td>
<td>1.211</td>
<td>1.196</td>
<td>1.232</td>
</tr>
</tbody>
</table>

VIII. CONCLUSION