2013 IEEE Congress on Evolutionary Computation
June 20-23, Cancún, México

A Parallel Genetic Algorithm with Edge Assembly Crossover for 100,000-City Scale TSPs

Kazuma Honda
Interdisciplinary Graduate School of Science and Engineering,
Tokyo Institute of Technology,
Yokohama, Japan
Email: khonda@ic.dis.titech.ac.jp

Yuichi Nagata
Education Academy of Computational Life Science,
Tokyo Institute of Technology,
Yokohama, Japan
Email: nagata@acls.titech.ac.jp

Isao Ono
Interdisciplinary Graduate School of Science and Engineering,
Tokyo Institute of Technology,
Yokohama, Japan
Email: isao@dis.titech.ac.jp

***Abstract*—In this paper, we propose a new parallel genetic algorithm (GA) with edge assembly crossover (EAX) for the traveling salesman problem (TSP). GA with EAX (GA-EAX) is one of the promising meta-heuristics for TSP and found best-known tours for several well-known 100,000-city scale TSP instances. However, it takes about ten days to execute this GA just one time using the default configuration on the 120,000-city instance [1]. Therefore, it is crucial to reduce the running time of GA-EAX for 100,000-city scale instances in order to make it possible to improve the algorithm through trial and error. The proposed parallel GA achieves about twenty-times speed up without deteriorating the quality of solutions compared to the original GA-EAX. We also demonstrate that the proposed parallel GA successfully finds new best-known tours for the 120,000-city and 180,000-city instances called vangogh120K and courbet180K, respectively.**

I. Introduction

The traveling salesman problem (TSP) is one of the typical combinatorial optimization problems. The objective of TSP is to find the shortest Hamilton cycle in a weighted complete graph. Since TSP is not only intuitive but also NP-hard, this problem is often used as an ideal platform for testing new meta-heuristics approaches. There are various benchmark instances of TSP. For example, Art TSPs (http://www.tsp.gatech.edu/data/art/index.html) consists of instances with sizes ranging from 100,000 to 200,000.

As far as we know, there are two promising approaches that have been applied to 100,000-city scale instances like Art TSPs. The first approach is based on the Lin-Kernighan (LK) algorithm [2] and the Iterated Local Search [3]. The Chained LK [4] and LKH-2 [5] are well-known implementations of this approach. LKH-2 is known as one of the most effective heuristic algorithms for finding very high-quality tours. The second approach is based on genetic algorithms (GAs). Hybrid GAs called memetic algorithms (MAs) that combine GAs with the LK algorithm have been studied [6]–[9]. [7] and [8] parallelize MA with QBX crossover and the LK algorithm and the hybrid GA based on the coarse-grained model [10], respectively. The GA with edge assembly crossover (GA-EAX) has shown good performance for up to 200,000-city instances [1]. It has been reported that GA-EAX finds better solutions than LKH-2 with shorter running time for almost instances [1].

GA-EAX consists of two stages called GA-EAX/Stage1 and GA-EAX/Stage2. GA-EAX/Stage1 employs a crossover operator called EAX-Single in order to improve population members (tours) while maintaining the population diversity as high as possible. EAX-Single exchanges edges locally between two parent tours. On the other hand, GA-EAX/Stage2 employs a crossover operator called EAX-Block2 in order to improve the population members obtained after invoking GA-EAX/Stage1. EAX-Block2 exchanges edges globally between two parent tours. GA-EAX was applied to Art TSPs and improved the best-known tours for all the instances. However, it takes about ten days to execute this GA just one time using the default configuration on the 120,000-city instance [1]. This means that it is almost impossible to develop an algorithm of GA-EAX suitable for 100,000-city scale instances through trial and error and to apply GA-EAX to more large-scale instances. Therefore, it is crucial to reduce the running time of GA-EAX for 100,000-city scale instances in order to make it possible to improve the algorithm through trial and error. We believe that the running time should be less than one day if we wish to develop an algorithm through trial and error. Since GA-EAX/Stage1 consumes 90% of the running time of GA-EAX, GA-EAX/Stage1 should be speeded up more than ten-times faster.

The master/worker model is a promising parallel model for speeding up an optimization method without performance degradation because it is not necessary to modify its search algorithm. The master/worker model can be applied to an optimization method if the optimization method includes procedures that are performed independently to each other and whose running time is much longer than that of the rest procedures [11]. For example, evaluation procedures are parallelized based on the master/worker model in existing parallel GAs [10], [12].

Most of the running time of GA-EAX/Stage1 is spent in the procedure for generating offspring solutions. In fact, this procedure can be easily parallelized because a number of parents' pairs generate offspring solutions independently. However, if we parallelize this procedure by the master/worker model, the total communication time between the master node and the worker nodes becomes longer than the running time required for generating offspring solutions because the data size of individuals that must be transferred is too large in 100,000-city scale instances. This means that the running time after parallelization becomes longer than that before parallelization. Therefore, it is difficult to speed up GA-

978-1-4799-0454-9/13/$31.00 ©2013 IEEE

EAX/Stage1 by parallelization without modifying the original search algorithm.

In this paper, we propose a new parallel GA with EAX-Single in order to reduce the running time to less than one day without deteriorating the quality of solutions in 100,000-city scale TSP instances. We demonstrate that the proposed parallel GA achieves about twenty-times speed up without deteriorating the solution quality of the original GA-EAX/Stage1 in all Art TSPs instances [1]. We also demonstrate that the proposed parallel GA successfully finds new best-known tours for the 120,000-city and 180,000-city instances called vangogh120K and courbet180K, respectively.

This paper is organized as follows. Section II introduces the algorithm of GA-EAX/Stage1. In section III, we consider parallelizing GA-EAX/Stage1 by the master/worker model without modifying the original algorithm and point out that the running time of the parallelized GA-EAX/Stage1 will become longer than that of the original implementation. In section IV, we propose a new parallel GA with EAX-Single. Section V compares the performance of the proposed parallel GA with that of the original GA-EAX/Stage1. Section VI is discussion in which we try to find new best-known tours of Art TSPs instances. Section VII concludes this paper.

II. Algorithm of GA-EAX/Stage1

GA-EAX consists of two phases: GA-EAX/Stage1 and GA-EAX/Stage2. In this paper, we consider to parallelize GA-EAX/Stage1 because its running time is about 90% of the total running time of GA-EAX. In this section, we briefly explain the crossover operator called EAX-Single and the generation alternation model used in GA-EAX/Stage1. We refer to the reader to the original paper [1] for more details.

A. EAX-Single

EAX generates more than one offspring solutions from a single pair of parents and EAX-Single is a variant of EAX.

Algorithm 1 shows the algorithm of EAX-Single, which generates an offspring solution y from a single pair of parents, $p_{\rm A}$ and $p_{\rm B}$. Then, $p_{\rm A}$ and $p_{\rm B}$ play a role of an acceptor and a donor, respectively. EAX-Single replaces some edges of $p_{\rm A}$ with those of $p_{\rm B}$. Let $E_{\rm A}$, $E_{\rm B}$ and E_y be sets of the edges of $p_{\rm A}$, $p_{\rm B}$ and y, respectively. After step 5, an obtained intermediate solution y often consists of several subtours and it is modified into a tour in step 6. In order to generate more than one offspring solutions from the same pair of parents, steps 3–6 are repeated, choosing a different AB-cycle in step 4.

B. Generation Alternation Model of GA-EAX/Stage1

Algorithm 2 shows the generation alternation model of GA-EAX/Stage1. Let $N_{\rm pop}$ and $N_{\rm kids}$ be the population size and the number of offspring solutions generated from a single pair of parents, $p_{\rm A}$ and $p_{\rm B}$, respectively. The values of $N_{\rm pop}$ and $N_{\rm kids}$ are 300 and 30, respectively, in the default configuration of GA-EAX/Stage1 [1]. The edge frequency table $F(e)$ is a table that records the frequencies of each edge $e \in E$ included in the population, where E is the edge set of the complete graph of a given TSP instance. The values of $F(e)$ $(e \in E)$ are initialized in step 1 and are used in the evaluation function for selecting offspring solutions in step 6. This evaluation function is based on the edge entropy measure computed from $F(e)$ and is used for maintaining the population diversity in a positive manner. Let y^* be the selected individual in step 5, which replaces the population member chosen as parent $p_{\rm A}$. The values of $F(e)$ are updated in step 6 as follows:

$$F(e) \leftarrow F(e) - 1 \quad \forall e \in E_{\rm remove},$$
$$F(e) \leftarrow F(e) + 1 \quad \forall e \in E_{\rm add},$$

where $E_{\rm remove}$ is a set of the edges that are included in $p_{\rm A}$ but not included in y^*, $E_{\rm add}$ is a set of the edges that are included in y^* but not included in $p_{\rm A}$.

In step 6 of Algorithm 2, the offspring y^* is selected, taking account of the balance between the amount of the improvement

Algorithm 1 EAX-Single

1: Generate an undirected multigraph defined as $G_{\rm AB} = (V, E_{\rm A} \cup E_{\rm B})$.
2: Extract AB-cycles from $G_{\rm AB}$ by repeating a procedure of traversing edges of $E_{\rm A}$ and ones of $E_{\rm B}$ alternatively in $G_{\rm AB}$ until an AB-cycle is obtained. Here, an AB-cycle is defined as a cycle in $G_{\rm AB}$, such that edges of $E_{\rm A}$ and ones of $E_{\rm B}$ are alternately linked.
3: Copy parent $p_{\rm A}$ to intermediate solution y: $y \leftarrow p_{\rm A}$.
4: Choose an AB-cycle from the set of AB-cycles randomly.
5: Generate an intermediate solution by removing the edges of $E_{\rm A}$ in the chosen AB-cycle from the intermediate solution y and adding those of $E_{\rm B}$ in the AB-cycle to y.
6: Connect all the subtours in the intermediate solution y to generate an offspring solution y (a single tour) by using a local search heuristic.

Algorithm 2 The generation alternation model of GA-EAX/Stage1

1: **(Initialization of Population)** Generate an initial population that consists of $N_{\rm pop}$ individuals by using a local search heuristics with the 2-opt operator. Then, initialize the number of generation, $g \leftarrow 0$, and the edge frequency table $F(e)$ so as to record the frequencies of each edge included in the initial population. Let the population P^g be the initial population.
2: **(Mating Selection)** Re-label the indices of the population members randomly. Let x_i be the ith individual in the population P^g $(i \in \{1, \ldots, N_{\rm pop}\})$.
3: **for** $i \leftarrow 1$ **to** $N_{\rm pop}$ **do**
4: $(p_{\rm A}, p_{\rm B}) \leftarrow (x_i, x_{i+1})$, where $x_{N_{\rm pop}+1} = x_1$.
5: **(Offspring Generation)** Generate offspring solutions $\{y_1, \ldots, y_{N_{\rm kids}}\}$ by EAX-Single($p_{\rm A}, p_{\rm B}$).
6: **(Survival Selection)** Select the best individual y^* from $\{y_1, \ldots, y_{N_{\rm kids}}, p_{\rm A}\}$ in terms of the evaluation function. Then, replace x_i $(= p_{\rm A})$ in P^g with the selected individual y^*. Update the edge frequency table $F(e)$ using x_i and y^*.
7: **end for**
8: **(Termination Condition Check)** If the termination condition described in section II-B is not satisfied, then $g \leftarrow g + 1$ and go to step 2.

and loss of the population diversity. Let L be the average tour length of the population and H the edge entropy of the population defined as follows:

$$H = -\sum_{e \in E} F(e)/N_{\mathrm{pop}}(\log(F(e)/N_{\mathrm{pop}})). \quad (1)$$

$\Delta L(y)$ and $\Delta H(y)$ denote the differences in L and H, respectively, when x_i (p_{A}) is replaced with y^* in step 6 of Algorithm 2. The offspring y^* is selected so that the following evaluation function is maximized.

$$\mathrm{Eval}_{\mathrm{Ent}}(y) := \begin{cases} \dfrac{\Delta L(y)}{\Delta H(y)} & (\Delta L < 0, \Delta H < 0) \\ -\dfrac{\Delta L(y)}{\epsilon} & (\Delta L < 0, \Delta H \geq 0), \\ -\Delta L(y) & (\Delta L \geq 0) \end{cases} \quad (2)$$

where y is an offspring solution and ϵ is a sufficiently small positive number.

GA-EAX/Stage1 is terminated when the best solution in the population does not improved for a predefined period of generations N_{stag}.

III. Difficulty in Parallelization of GA-EAX/Stage1

In this section, we consider to parallelize GA-EAX/Stage1 by the master/worker model [11] in order to reduce its running time without deteriorating the original performance. The master/worker model enables us to parallelize GA-EAX/Stage1 without changing the original search algorithm, but this parallelization has a drawback as described later.

In section III-A, we measure the running time of each procedure of GA-EAX/Stage1 to identify which procedure occupies the largest portion of the total running time. In section III-B, we design a master/worker model of GA-EAX/Stage1 in which the procedures identified in section III-A are executed in parallel on worker nodes to speed up GA-EAX/Stage1 without modifying the original search algorithm. Then, we point out that the running time of GA-EAX/Stage1 parallelized by this master/worker model becomes longer than that of the original GA-EAX/Stage1 running on only a single node.

A. Measuring Running Time

We measure the running time of each procedure in GA-EAX/Stage1 to identify procedures that have large portion of the total running time. Since our objective is to parallelize GA procedures, we do not measure the running time of step 1 of Algorithm 2.

We use the 120,000-city instance named vangogh120K from the Art TSPs for benchmarking. We use the default configuration for setting the parameters of GA-EAX/Stage1: $N_{\mathrm{pop}} = 300$ and $N_{\mathrm{kids}} = 30$. According to this configuration, 300×30 offspring solutions are generated in each generation where one generation refers to one iteration of steps 2–8 of Algorithm 2. We execute GA-EAX/Stage1 on the environment shown in Appendix A.

The measured running time of each procedure is as follows:

- The running time of mating selection (Step 2): less than 1 [msec]
- The running time of offspring generation (Step 5): $10,596$ [msec]
- The running time of survival selection (Step 6): 88 [msec]
- The total running time of one generation: $10,684$ [msec]

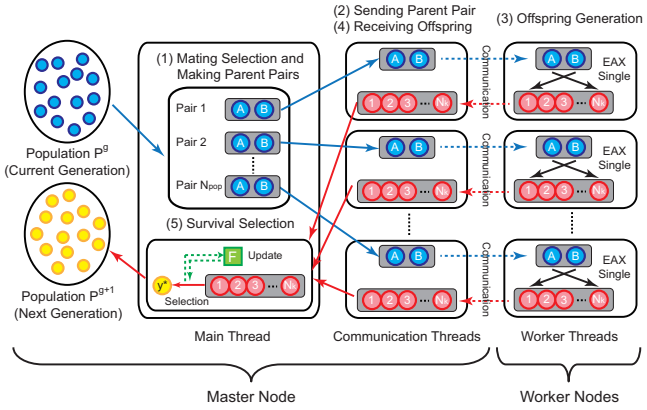

Fig. 1. Diagram of parallelized GA-EAX/Stage1 based on the master/worker model. The blue and yellow circles are the population members of the current and next generations, respectively. The red circles are offspring solutions generated by EAX-Single. The green square is the edge frequency table F. N_{k} corresponds to N_{kids} in Algorithm 2.

In the next section, we discuss the effect of the parallelization by the master/worker model based on the obtained results.

B. Parallelizing GA-EAX/Stage1 by Master/Worker Model

As shown in steps 4 and 5 of Algorithm 2, the procedure of offspring generation can be executed independently for each pair of parents $(p_{\mathrm{A}}, p_{\mathrm{B}}) = (x_i, x_{i+1})$, $i \in \{1, \ldots, N_{\mathrm{pop}}\}$. On the other hand, the procedure of survival selection in step 6 cannot be executed independently for each pair of parents because the edge frequency table F has to be updated every time individual x_i (selected as parent p_{A}) in the population is replaced with the selected individual y^*. Therefore, we consider a master/worker model in which multiple worker nodes execute the procedure of offspring generation in parallel and the master node executes the procedures of mating selection and survival selection as shown in Fig. 1. The main thread and multiple communication threads work cooperatively on the master node. Each communication thread communicates with the corresponding worker node.

The master node sends a pair of parents to each worker node and each worker node returns all the generated offspring solutions to the master node. It takes 1.25 [msec] to send an individual between two nodes in the environment described in the previous section. Hence, in the default configuration of GA-EAX/Stage1, the total communication time from the master node to the worker nodes per generation is

$$1.25 \times 300 \times 2 = 750 \text{ [msec]}$$

since the master node must send N_{pop} ($= 300$) pairs of parents (two individuals) to the worker nodes. On the other hand, the

total communication time from the worker nodes to the master node is

$$1.25 \times 300 \times 30 = 11,250 \text{ [msec]}$$

because the master node must receive N_{kids} $(= 30)$ offspring solutions from each worker node. Thus, the total communication time is $12,000$ [msec] per generation. Note that the total communication time does not depend on the number of worker nodes. Therefore, it is difficult to reduce the running time by parallelizing the procedure of GA-EAX/Stage1 using this master/worker model because the communication time per one generation ($12,000$ [msec]) is longer than the total running time of one generation before parallelization ($10,684$ [msec]).

IV. Proposed Parallel GA with EAX-Single

Generally, parallelization based on the master/worker model can reduce the total running time if the running time on each worker node is sufficiently longer than the communication time between the master node and worker nodes. As described in the previous section, however, the total running time of GA-EAX/Stage1 will increase by the parallelization based on the straightforward master/worker model because the communication time between the master node and worker nodes dominates the overall running time on 100,000-city scale instances.

In this section, we propose a new parallel GA based on a more suitable master/worker model in order to reduce the total running time of GA-EAX/Stage1. The proposed parallel GA employs a new generation alternation model, which is slightly different from the original one, in order to make it possible to reduce the total running time by the parallelization based on the master/worker model. In section IV-A, we first discuss requirements for a new parallel GA. Next, we propose a new parallel GA to achieve the requirements in section IV-B. Finally, we describe the detailed algorithms of the proposed parallel GA in section IV-C.

A. Requirements for Proposed Parallel GA

In order to reduce the total running time by parallelization based on the master/worker model, the running time on each worker node should be sufficiently longer than the communication time between the master node and worker nodes. However, the total running time after parallelization will increase if the running time on each worker node is just increased. Thus, we have to reduce the number of generations (one generation is defined as the replacement of all individuals on the master node) required for convergence simultaneously in order to reduce the total running time. In order to reduce the total running time by parallelization, we believe that the proposed parallel GA should meet the following four requirements.

Requirement 1: The running time on each worker node should be sufficiently longer than that on the master node and the communication time between the master node and worker nodes.

Requirement 2: The number of generations required for convergence in a new parallel GA should be smaller than that in GA-EAX/Stage1.

Requirement 3: Tour length obtained by the proposed parallel GA should be equal to or shorter than that by GA-EAX/Stage1.

Requirement 4: Tour length obtained by the combination of the proposed parallel GA and GA-EAX/Stage2 should be equal to or shorter than that by the combination of GA-EAX/Stage1 and GA-EAX/Stage2.

B. Proposed Parallel GA with EAX-Single

In this section, we propose a new parallel GA with EAX-Single that meets the four requirements discussed in the previous section.

Fig. 2 shows a diagram of the proposed parallel GA with EAX-Single based on the master/worker model. As shown in Fig. 2, the master node executes mating selection and update of the master edge frequency table F^{master}. On the other hand, each worker node does both offspring generation, survival selection and update of the worker edge frequency table F_i^{worker}, which is a copy of F^{master} at the beginning of each generation, where i is the index of each worker.

For Requirement 1, each individual in the chunk generates offspring solutions with multiple other individuals on a worker thread as shown in Fig. 2 (3). In the master/worker model of GA-EAX/Stage1 described in section III-B, each individual in the population generates offspring solutions with only a single other individual in the population on each worker thread as shown in Fig. 1. This is a reason that the running time per generation on a worker node is short. On the other hand, in the proposed parallel GA, each individual generates offspring solutions with multiple individuals in one generation. Thus, the running time per generation on each worker node in the proposed parallel GA becomes longer than that in the master/worker model of GA-EAX/Stage1.

The proposed parallel GA is expected to meet Requirement 2 because each individual generates offspring solutions with multiple other individuals as shown in Fig. 2 (3). In the master/worker model of GA-EAX/Stage1, the number of replaced edges per generation in each individual tends to be very small, which means that improvement of the tour length per generation is likely to be small. This is caused by two reasons. First, GA-EAX/Stage1 replaces edges of each individual chosen as an acceptor (p_{A} in Algorithm 1) in the population with ones from only a single other individual chosen as a donor (p_{B} in Algorithm 1) by EAX-Single in one generation. Second, EAX-Single replaces edges in the acceptor with ones of only a single AB-cycle generated with the acceptor and the donor. On the other hand, the proposed parallel GA replaces edges of each individual chosen as an acceptor in the chunk with ones from all the other individuals chosen as donors in the chunk. This means that the number of replaced edges per generation in each individual becomes large. Thus, improvement of the tour length per generation in the proposed parallel GA is expected to be larger than that in the master/worker model of GA-EAX/Stage1. By this property of the proposed parallel GA, we expect the proposed parallel GA to achieve Requirement 3 and 4.

In order to achieve Requirement 3 and 4, we also redesign a termination condition. We cannot use the same termination condition as that of GA-EAX/Stage1 because the convergence speed of the population should depend on how many times survival selection is applied to each individual in the population. The number of survival selection to an individual per

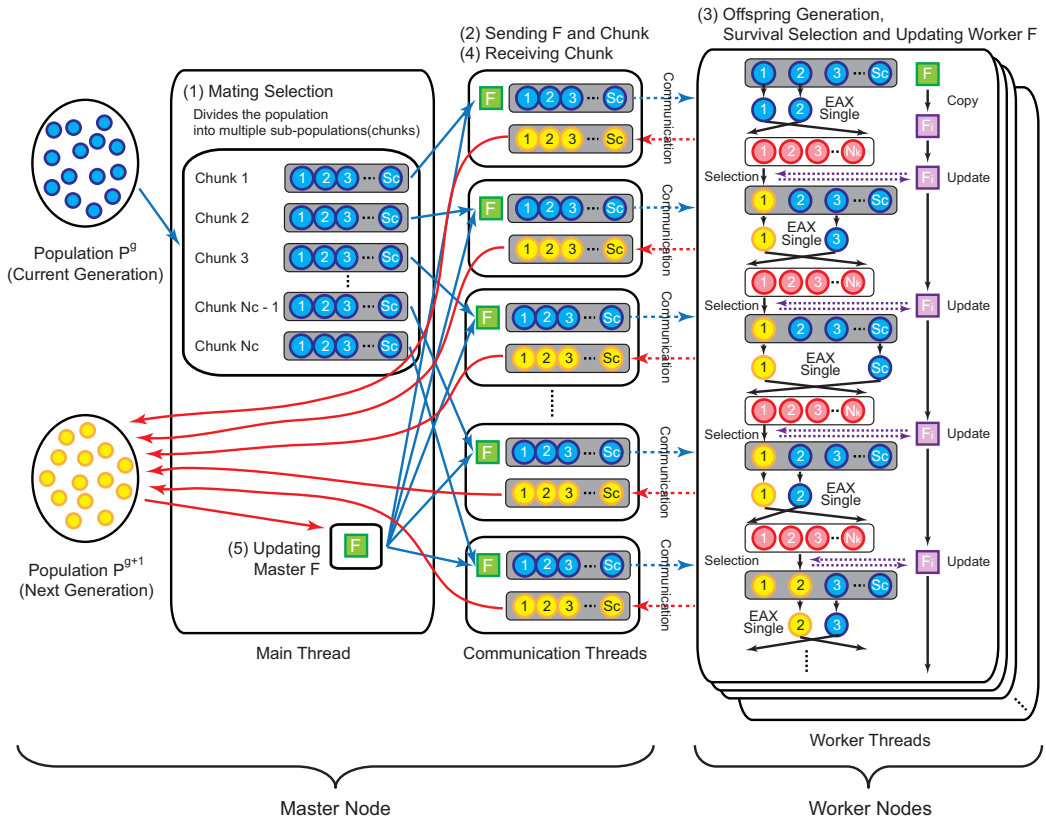

Fig. 2. Diagram of the proposed parallel GA based on the master/worker model. Blue and yellow circles are the population members of the current and next generations, respectively. Red circles are offspring solutions generated by EAX-Single. Green and purple squares are the master and each worker edge frequency tables, respectively. $N_{\mathrm{k}}, N_{\mathrm{c}}$ and S_{c} correspond to $N_{\mathrm{kids}}, N_{\mathrm{chunk}}$ and S_{chunk}, respectively, in Algorithm 3 and 4.

generation in the proposed parallel GA is $S_{\mathrm{chunk}} - 1$ times more than that in GA-EAX/Stage1. As shown in Fig. 2 (3), the proposed parallel GA applies survival selection to Individual 1 $S_{\mathrm{chunk}} - 1$ times per generation on each worker thread. On the other hand, as shown in Fig. 1, GA-EAX/Stage1 applies survival selection to Parent A only one time per generation on each worker. Note that Parent A in Fig. 1 corresponds to Individual 1 in Fig. 2. Thus, in the proposed parallel GA, we employ $N_{\mathrm{stag}}/(S_{\mathrm{chunk}} - 1)$ as the predefined period of generations of the termination condition while N_{stag} is used in GA-EAX/Stage1.

The flow of procedures executed in one generation in the proposed parallel GA with EAX-Single is as follows:

(1) Mating selection on the master node: The main thread on the master node shuffles the population randomly and divides the population members of the current generation into multiple equal sub-populations called "chunks" as shown in Fig. 2 (1).

(2) Sending chunks and master edge frequency table on the master node: Each communication thread on the master node sends a chunk and a copy of the master edge frequency table F^{master} to the corresponding worker thread as shown in Fig. 2 (2).

(3) Offspring generation, survival selection and updating worker edge frequency table on each worker node: The worker thread on each worker node receives a chunk and F^{master} from the corresponding communication thread on the master node and copies F^{master} to a worker edge frequency table F_i^{worker}. Then, the worker thread repeats offspring generation, survival selection and update of F_i^{worker} as follows. As shown in Fig. 2 (3), the worker thread chooses Individual 1, the blue circled 1, as an acceptor and Individual 2, the blue circled 2, as a donor from the chunk, respectively, and applies EAX-Single to them to make N_{kids} offspring solutions, the red circles, and selects the best individual in terms of the evaluation function described in section II-B, the yellow circle. Note that F_i^{worker} is used as an edge frequency table in the evaluation function. Then, it replaces the acceptor in the chunk with the selected individual and updates F_i^{worker}. Note that the selected individual becomes Individual 1, i.e. the next acceptor. After finishing the combination of Individual 1 as an acceptor and Individual 2 as a donor, the worker thread applies offspring generation, survival selection and update of F_i^{worker} to the acceptor, i.e. Individual 1, changing the donor from Individual 3 to Individual S_{c} as shown in Fig. 2 (3).

(4) Receiving chunks on the master node: Each communication thread on the master node receives only a chunk from the corresponding worker thread and adds the individuals in the chunk to the population of the next generation.

(5) Updating master edge frequency table on the master node: After the population of the next generation is completed, the main thread on the master node updates the master edge frequency table F^{master} based on the population of the next generation. Note that each worker thread holds its own

Algorithm 3 The proposed parallel GA running on the master node

1: **(Initialization of Population)** Generate an initial population that consists of N_{pop} individuals by using a local search heuristics with the 2-opt operator. Then, initialize the number of generation, $g \leftarrow 0$, and the master edge frequency table F^{master} so as to record the frequencies of each edge included in the initial population. Let the population P^g be the initial population.
2: **(Mating Selection)** Shuffle the population P^g randomly and divide the population P^g into N_{chunk} sub-populations called "chunks". Let $x_{i,j}$ be the jth individual of the ith chunk ($i \in \{1, \ldots, N_{\mathrm{chunk}}\},\ j \in \{1, \ldots, S_{\mathrm{chunk}}\}$). Note that $N_{\mathrm{pop}} = N_{\mathrm{chunk}} \times S_{\mathrm{chunk}}$.
3: **for** $i \leftarrow 1$ **to** N_{chunk} **in parallel do**
4: Send the ith chunk $\{x_{i,1}, \ldots, x_{i,S_{\mathrm{chunk}}}\}$ and a copy of the master edge frequency table F^{master} to the ith worker node
5: Receive the ith chunk from the ith worker node and add the chunk to the next population P^{g+1}.
6: **end for**
7: Update the master edge frequency table F^{master} based on the next population P^{g+1}.
8: **(Termination Condition Check)** If the termination condition described in section IV-B is not satisfied, then $g \leftarrow g + 1$ and go to step 2.

edge frequency table F_i^{worker} and, thus, can execute survival selection independently to each other simultaneously.

C. Details of Algorithms Running on Master Node and Each Worker Node

Algorithm 3 shows the algorithm of the proposed parallel GA running on the master node. In Algorithm 3, steps 1, 2, 7 and 8 are executed by the master thread and steps 3–6 are done by the communication threads in Fig. 2. Algorithm 4 describes the algorithm of the proposed parallel GA running on each worker node.

V. Experiments

In this section, we confirm the following two points through numerical experiments using 100,000-city scale benchmark instances.

1) The running time of the proposed parallel GA is more than ten-times faster than that of GA-EAX/Stage1 and the tour length obtained by the proposed parallel GA is equal to or shorter than that by GA-EAX/Stage1.
2) Tour length obtained by the combination of the proposed parallel GA and GA-EAX/Stage2 is equal to or shorter than that by the combination of GA-EAX/Stage1 and GA-EAX/Stage2.

A. Benchmark Instances

We use all Art TSPs instances called mona-Lisa100K, vangogh120K, venus140K, pareja160K, courbet180K and earring200K. Note that the suffix number of each instance name represents the number of cities of the instance.

Algorithm 4 The proposed parallel GA running on each worker node

1: Assume that this worker node is the ith one. Receive the ith chunk $\{x_{i,1}, \ldots, x_{i,S_{\mathrm{chunk}}}\}$ and the master edge frequency table F^{master} from the master node. Then, copy it to the worker edge frequency table: $F_i^{\mathrm{worker}} \leftarrow F^{\mathrm{master}}$.
2: **for** $j \leftarrow 1$ **to** S_{chunk} **do**
3: **for** $k \leftarrow 1$ **to** $S_{\mathrm{chunk}} - 1$ **do**
4: $(p_{\mathrm{A}}, p_{\mathrm{B}}) \leftarrow (x_{i,j}, x_{i,j+k})$ for all $l \in \{1, \ldots, S_{\mathrm{chunk}}\}$.
5: **(Offspring Generation)** Generate offspring solutions $\{y_1, \ldots, y_{N_{\mathrm{kids}}}\}$ by EAX-Single$(p_{\mathrm{A}}, p_{\mathrm{B}})$.
6: **(Survival Selection)** Select the best individual from $\{y_1, \ldots, y_{N_{\mathrm{kids}}}, p_{\mathrm{A}}\}$ in terms of the evaluation function. Then, replace $x_{i,j}$ $(= p_{\mathrm{A}})$ with the selected individual y^*. Update the worker edge frequency table F_i^{worker} using x_i and y^*.
7: **end for**
8: **end for**
9: Send the ith chunk to the master node.
10: If the program on the master node is not terminated, then go to step 1.

B. Performance Indices

The performance indices used in the experiments are the average running time [hours], the average speed up rate over three trials and the relative error in tour length [%]. The speed up rate is given by the running time of GA-EAX/Stage1 divided by that of the proposed parallel GA. The relative error in tour length is defined as follows:

$$\text{Relative Error} = \frac{l_{\mathrm{best}} - l_{\mathrm{opt}}}{l_{\mathrm{opt}}} \times 100\ [\%], \tag{3}$$

where l_{best} is tour length of the best individual in the population and l_{opt} is that of the best-known tour of each instance.

C. Experimental Configurations

The proposed parallel GA and GA-EAX/Stage1 are run three-times on all benchmark instances. All the programs are executed on the supercomputer TSUBAME 2.0 installed at Tokyo Institute of Technology. Please see Appendix A for more details of the environment. Configurations for each algorithm are follows.

GA-EAX/Stage1: We set the population size, $N_{\mathrm{pop}} = 300$ and the number of offspring solutions generated from a single pair of parents, $N_{\mathrm{kids}} = 30$, which are the default configurations of GA-EAX/Stage1 [1]. GA-EAX/Stage1 is executed on only a single node of the environment.

Proposed parallel GA: We set $N_{\mathrm{pop}} = 300$ and $N_{\mathrm{kids}} = 30$, which are the same configuration of GA-EAX/Stage1. We also set the number of individuals in a chunk, $S_{\mathrm{chunk}} = 10$ and the number of the total chunks, $N_{\mathrm{chunk}} = 30$. The proposed parallel GA is executed on total 31 nodes so that each of the master node and worker nodes is assigned to one node. Note that a single core is used in each worker node, which means that 30 cores in total are used for the worker nodes.

GA-EAX/Stage2: We use the default configuration of GA-EAX/Stage2. Please see original paper [1] for details.

TABLE I. THE AVERAGE RUNNING TIME OF GA-EAX/STAGE1 AND THE PROPOSED PARALLEL GA AND THE AVERAGE SPEED UP RATE WHEN THE RELATIVE ERROR REACHES 1.0, 0.1 AND 0.01% AND EACH ALGORITHM IS TERMINATED. "INSTANCE" INDICATES THE INSTANCE NAME. "GA-EAX" AND "PROPOSED" INDICATE THE AVERAGE RUNNING TIME [HOURS] OF GA-EAX/STAGE1 AND THE PROPOSED PARALLEL GA, RESPECTIVELY. "S-UP" INDICATES THE AVERAGE SPEED UP RATE.

	Relative Error = 1 [%]			Relative Error = 0.1 [%]			Relative Error = 0.01 [%]			Terminate		
Instance	GA-EAX	Proposed	S-UP	GA-EAX	Proposed	S-UP	GA-EAX	Proposed	S-UP	GA-EAX	Proposed	S-UP
mona-lisa100K	12.8	0.6	×20.2	28.3	1.4	×20.1	65.6	3.0	×22.2	114.4	4.8	×23.8
vangogh120K	20.7	1.1	×19.2	44.4	2.3	×19.8	104.2	5.4	×19.3	179.1	8.4	×21.4
venus140K	28.1	1.4	×19.6	63.1	3.1	×20.1	132.5	6.8	×19.4	254.1	12.2	×20.8
pareja160K	41.9	2.1	×19.8	88.3	4.4	×20.2	216.4	10.6	×20.4	409.4	17.3	×23.6
courbet180K	58.3	2.8	×20.6	120.6	5.7	×21.1	302.8	14.1	×21.4	486.8	22.5	×21.6
earring200K	75.6	3.6	×20.8	147.2	7.1	×20.8	422.2	19.8	×21.4	676.0	29.8	×22.7

TABLE II. THE QUALITY OF THE FINAL SOLUTIONS OBTAINED BY GA-EAX/STAGE1 AND THE PROPOSED PARALLEL GA. "INSTANCE" INDICATES THE INSTANCE NAME. "GA-EAX/STAGE1" AND "PROPOSED GA" MEAN THE RESULTS OF GA-EAX/STAGE1 AND THE PROPOSED PARALLEL GA, RESPECTIVELY. "BEST [%]" AND "AVG [%]" ARE THE BEST RELATIVE ERROR IN THE THREE TRIALS AND THE AVERAGE RELATIVE ERROR OVER THE THREE TRIALS, RESPECTIVELY, WHEN EACH ALGORITHM IS TERMINATED.

	GA-EAX/Stage1		Proposed GA	
Instance	Best [%]	Avg [%]	Best [%]	Avg [%]
mona-lisa100K	0.00610	0.00661	0.00629	0.00673
vangogh120K	0.00645	0.00690	0.00737	0.00760
venus140K	0.00621	0.00662	0.00617	0.00692
pareja160K	0.00676	0.00700	0.00730	0.00772
courbet180K	0.00726	0.00743	0.00762	0.00778
earring200K	0.00797	0.00811	0.00749	0.00821

TABLE III. THE QUALITY OF THE FINAL SOLUTIONS OBTAINED BY COMBINING GA-EAX/STAGE1 WITH GA-EAX/STAGE2 AND THOSE BY COMBINING THE PROPOSED PARALLEL GA WITH GA-EAX/STAGE2. "INSTANCE" INDICATES THE INSTANCE NAME. "GA-EAX/STAGE1" AND "PROPOSED GA" MEAN THE RESULTS OF THE COMBINATION OF GA-EAX/STAGE1 AND GA-EAX/STAGE2 AND THE COMBINATION OF THE PROPOSED PARALLEL GA AND GA-EAX/STAGE2, RESPECTIVELY. "BEST [%]" AND "AVG [%]" ARE THE BEST RELATIVE ERROR IN THE THREE TRIALS AND THE AVERAGE RELATIVE ERROR OVER THE THREE TRIALS, RESPECTIVELY, WHEN EACH ALGORITHM IS TERMINATED.

	GA-EAX/Stage1		Proposed GA	
Instance	Best [%]	Avg [%]	Best [%]	Avg [%]
mona-lisa100K	0.00000	0.00008	0.00002	0.00006
vangogh120K	0.00017	0.00019	0.00011	0.00016
venus140K	0.00010	0.00017	0.00007	0.00015
pareja160K	0.00008	0.00011	0.00008	0.00010
courbet180K	0.00010	0.00013	0.00005	0.00008
earring200K	0.00011	0.00020	0.00021	0.00022

D. Results

TABLE I shows the average running time of GA-EAX/Stage1 and the proposed parallel GA and the speed up rate when the relative error reaches 1.0, 0.1 and 0.01% and each algorithm is terminated. As shown in TABLE I, the proposed parallel GA achieves about twenty-times speed up over GA-EAX/Stage1 at any relative error on all the benchmark instances. Parallel efficiency (the rate of the speed up to the number of cores used in all worker nodes) is about $20/30 \approx 67\%$. On 100,000 to 180,000-city scale instances, the running time of the proposed parallel GA is less than one day. TABLE II shows the best relative error in the three trials and the average relative error over the three trials when each algorithm is terminated. Comparing results of both algorithms in TABLE II, the quality of solutions obtained by the proposed parallel GA is almost equal to that by GA-EAX/Stage1 on all the benchmark instances.

TABLE III shows results of the best relative error in the three trials and the average relative error over the three trials obtained by combining the proposed parallel GA with GA-EAX/Stage2 and those by combining GA-EAX/Stage1 with GA-EAX/Stage2. In TABLE III, we can see that the combination of the proposed parallel GA and GA-EAX/Stage2 succeeded in finding as good tours as the combination of GA-EAX/Stage1 and GA-EAX/Stage2 while the proposed parallel GA is twenty-times faster than GA-EAX/Stage1 as shown in TABLE I.

VI. DISCUSSIONS

A. Comparison of Search Performance in terms of the Number of Evaluations

In this section, we compare the search performance of the proposed parallel GA with that of GA-EAX/Stage1 in terms of the number of evaluations, i.e. the accumulated number of generated offspring solutions. Fig. 3 shows graphs of the number of evaluations versus the relative error of both algorithms. In each graph, the red and blue curves are the results of the three trials of the proposed parallel GA and GA-EAX/Stage1, respectively. This result suggests that the search performance of the proposed parallel GA is almost the same as that of GA-EAX/Stage1 in terms of the number of evaluations.

B. Finding New Best-Known Tours

In this section, we try to find new best-known tours of Art TSPs instances.

The original paper [1] reportedly improved best-known tours of Art TSPs instances by the following procedure:

1) Execute GA-EAX/Stage1 and GA-EAX/Stage2 ten times.
2) Construct an initial population by assembling top 30 tours from each of the ten populations obtained in step 1.
3) Execute GA-EAX/Stage2 starting from the initial population constructed in step 2.

In the above procedure, GA-EAX/Stage1 and GA-EAX/Stage2 are executed with the default configuration where the initial population has $30 \times 10 = 300$ tours.

We tried to find new best-known tours of Art TSPs instances with the same procedure except that GA-EAX/Stage1 is replaced with the proposed parallel GA. TABLE IV shows tour lengths of the current best-known tours and those found by the above procedure with the proposed parallel GA. As shown in TABLE IV, we succeeded in finding new improved best-known tours of vangogh120K and courbet180K. We also found the current best-known tours of all other instances.

VII. Conclusion

In this paper, we proposed a new parallel GA with EAX-Single. We demonstrated that the proposed parallel GA achieves about twenty-times speed up without deteriorating the quality of solutions compared to GA-EAX/Stage1 through numerical experiments on 100,000-city scale instances. Furthermore, we successfully found new best-known tours of 120,000-city and 180,000-city instances by combining the proposed parallel GA with GA-EAX/Stage2.

As future work, we will try to improve best-known tours of other 100,000-city scale instances and to apply the proposed parallel GA to more large scale instances like the World TSP (http://www.tsp.gatech.edu/world).

References

[1] Y. Nagata and S. Kobayashi, "A Powerful Genetic Algorithm Using Edge Assembly Crossover for the Traveling Salesman Problem," *INFORMS Journal on Computing (in press)*, 2012, published as an article in advance.

[2] S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[3] D. S. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization," in *Local Search in Combinatorial Optimization*. John Wiley and Sons, Ltd, 1997, pp. 215–310.

[4] D. Applegate, W. Cook, and A. Rohe, "Chained Lin-Kernighan for Large Traveling Salesman Problems," *INFORMS Journal on Computing*, vol. 15, no. 1, pp. 82–92, 2003.

[5] K. Helsgaun, "General k-opt submoves for the Lin-Kernighan TSP heuristic," *Mathematical Programming Computation*, vol. 1, no. 2, pp. 119–163, 2009.

[6] P. Merz and B. Freisleben, "Memetic Algorithms for the Traveling Salesman Problem," *Complex System*, vol. 13, no. 4, pp. 297–345, 2001.

[7] P. Merz and T. Fischer, "A Memetic Algorithm for Large Traveling Salesman Problem Instances," in *7th Metaheuristics International Conference*, 2007.

[8] H. D. Nguyen, I. Yoshihara, K. Yamamori, and M. Yasunaga, "Implementation of an Effective Hybrid GA for Large-Scale Traveling Salesman Problems," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 37, no. 1, pp. 92–99, 2007.

[9] M. Kuroda, K. Yamamori, M. Munetomo, M. Yasunaga, and I. Yoshihara, "A Proposal for Zoning Crossover of Hybrid Genetic Algorithms for Large-scale Traveling Salesman Problems," in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–6.

[10] E. Cantú-Paz, "A Survey of Parallel Genetic Algorithms," *IlliGAL Report No. 97007*, 1997.

[11] T. G. Sanders, B. A. Massingill, and B. L. Mattson, *Patterns for Parallel Programming*. Addison-Wesley Professional, 2004.

[12] H. Imade, R. Morishita, I. Ono, N. Ono, and M. Okamoto, "A Grid-Oriented Genetic Algorithm Framework for Bioinformatics," *New Generation Computing - Grid Systems for Life Sciences*, vol. 22, no. 2, pp. 177–186, 2004.

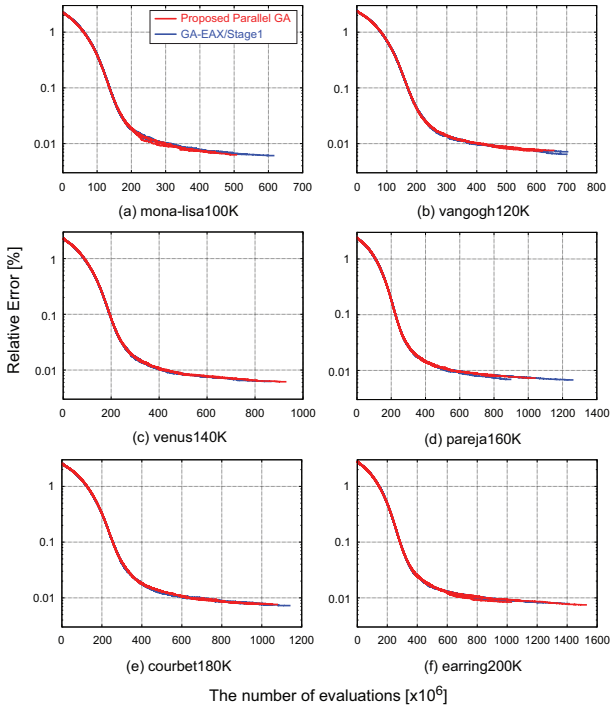

Fig. 3. The number of evaluations (the accumulated number of generated offspring) versus the relative error of GA-EAX/Stage1 and the proposed parallel GA.

TABLE IV. Results of trying to find new best-known tours. "Instance", "Best-Known" and "Proposed GA" indicate the instance name, tour length of the current best-known tour and that obtained by the procedure of section VI-B with the proposed parallel GA. The proposed parallel GA succeeded in finding new best known tours of vangogh120K and courbet180K.

Instance	Best-Known	Proposed GA
mona-lisa100K	5757191	5757191
vangogh120K	6543610	**6543609**
venus140K	6810665	6810665
pareja160K	7619953	7619953
courbet180K	7888733	**7888731**
earring200K	8171677	8171677

Appendix A
Computational Environment

In this paper, we conducted all the numerical experiments on the supercomputer TSUBAME 2.0 installed at Tokyo Institute of Technology. The details of the environment are follows:

CPU Intel Westmere-EP 2.93GHz $\times$ 2,
Cores 6×2,
Memory DDR3-1333 54GB,
Network QDR Infiniband (40Gbps),
OS SUSE Linux Enterprise Server 11 SP1,
Java Oracle JDK 1.7.0_07.

We implemented all the programs in Java and employed NIO-2 library for communication. Furthermore, in order to speed up data transfer, we used Socket Direct Protocol that enables us to utilize Remote Direct Memory Access over the Infiniband network as a communication protocol.