PAPER Special Section on Parallel and Distributed Computing and Networking

Accelerating the Held-Karp Algorithm for the Symmetric Traveling Salesman Problem

Kazuro KIMURA[†a)], **Shinya HIGA**[†], ***Nonmembers***, **Masao OKITA**[††], ***and*** **Fumihiko INO**[††], ***Members***

SUMMARY In this paper, we propose an acceleration method for the Held-Karp algorithm that solves the symmetric traveling salesman problem by dynamic programming. The proposed method achieves acceleration with two techniques. First, we locate data-independent subproblems so that the subproblems can be solved in parallel. Second, we reduce the number of subproblems by a meet in the middle (MITM) technique, which computes the optimal path from both clockwise and counterclockwise directions. We show theoretical analysis on the impact of MITM in terms of the time and space complexities. In experiments, we compared the proposed method with a previous method running on a single-core CPU. Experimental results show that the proposed method on an 8-core CPU was 9.5–10.5 times faster than the previous method on a single-core CPU. Moreover, the proposed method on a graphics processing unit (GPU) was 30–40 times faster than that on an 8-core CPU. As a side effect, the proposed method reduced the memory usage by 48%.

key words: *symmetric traveling salesman problem, Held-Karp algorithm, parallelization, meet in the middle, GPU*

1. Introduction

The traveling salesman problem (TSP) is a problem that finds the minimum Hamiltonian cycle for a weighted complete graph $G = (V, E)$, where V and E are the sets of vertices and edges, respectively. A Hamiltonian cycle here is a cycle that visits each vertex exactly once and the minimum Hamiltonian cycle has the lowest-weight cycle, i.e., the lowest sum of edge weights, in the given graph G. Hereafter, we call the minimum Hamiltonian cycle *the optimal path*. We also call the weight of the optimal path *the optimal value*.

One special case of the TSP is *the symmetric TSP (sTSP)*, which deals with an undirected graph. That is, the given graph has the same weight between two vertices in each opposite direction: $w(i, j) = w(j, i)$ holds for all vertices $i, j \in V$, where $w(i, j)$ represents the weight of the edge $(i, j) \in E$ from vertices i to j. Many practical applications include the sTSP as an underlying fundamental problem. For example, welding robots can minimize their power consumption by solving the sTSP [1]. In more detail, an optimal motion plan can be obtained by solving the sTSP for a graph in which vertices represent the welding points and edge weights represent the costs required for moving between the welding points.

Algorithms for the sTSP can be classified into three approaches: exact, approximate, and heuristic approaches. Exact algorithms in [2], [3] find the optimal path but the problem size that can be solved in a reasonable time is limited due to high demand for both compute and memory capacities. By contrast, approximate and heuristic algorithms [4], [5] rapidly find an approximate path but the path is not optimal. Consequently, approximate and heuristic algorithms are typically used if the given graph has many vertices or the approximation error is acceptable for the overlying application. Exact algorithms are preferred otherwise. The latter case is of our interest where applications require an exact solution for the sTSP.

The Held-Karp algorithm [2] is an exact algorithm that deploys dynamic programming for the sTSP. To the best of our knowledge, the Held-Karp algorithm is the fastest exact algorithm [6] and its worst-case time complexity is $O(n^2 2^n)$, where n is the number of vertices. Therefore, accelerating the Held-Karp algorithm increases the problem size n that can be solved in a reasonable time. The original paper [2] is a theoretical paper that presented the Held-Karp algorithm with its recursive equations and complexity analysis. By contrast, Kubo [7] presented a real implementation with specific data and loop structures. The contribution of [7] over [2] are twofold: the paper (1) shows a data structure that represents a set of vertices with a bit sequence; and (2) finds a loop structure that correctly solves data-dependent subproblems according to sorted bit sequences.

In this paper, we propose an acceleration method for the Held-Karp algorithm, which solves the sTSP. The proposed method achieves acceleration with two techniques, parallelization and meet in the middle (MITM). The former identifies data-independent subproblems from those generated by the Held-Karp algorithm. On the other hand, the latter approximately halves the number of subproblems by exploiting the symmetric attribute embedded in the given graph. Moreover, the proposed method reduces the memory usage as a side effect. We also show theoretical analysis for understanding the impact of MITM in terms of the time and space complexities. The source code of our implementation is available at http://www-ppl.ist.osaka-u.ac.jp/research/code/.

The paper is structured as follows. Section 2 introduces related work to clarify the contribution of the present paper. Section 3 presents an overview of the Held-Karp algorithm

Manuscript received January 7, 2019.
Manuscript revised March 22, 2019.
Manuscript publicized August 23, 2019.

[†]The authors are with the School of Engineering Science, Osaka University, Toyonaka-shi, 560–8531 Japan.
[††]The authors are with the Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871 Japan.
a) E-mail: u779795i@alumni.osaka-u.ac.jp

DOI: 10.1587/transinf.2019PAP0008

with [7]. Section 4 describes the proposed method with theoretical analysis. Section 5 shows experiments that evaluate the performance of the proposed method. Finally, Sect. 6 concludes the paper and discusses future work.

2. Related Work

Concorde [3] is an exact sTSP solver that relies on a cutting-plane method. This solver is capable of finding the optimal path for a large graph of $n = 85,900$. However, its execution time depends not only on the number n of vertices, i.e., the problem size, but also on the weight of edges. In fact, Ahammed *et al.* [8] found that the execution time differed by 30,000 times though n was fixed. On the other hand, the execution time of the proposed method depends only on the number n of vertices. Therefore, the proposed method is useful for strict real-time situations where the optimal path must be computed within a certain period of time.

Moffat [9] parallelized the Held-Karp algorithm for the sTSP using a JavaScript application programming interface (API). Similar to the proposed method, this method exploited the data parallelism inherent in the Held-Karp algorithm. However, the maximum problem size was limited by $n \leq 16$ because the method simply stored subproblems in $O(n2^n)$ space. By contrast, the proposed method saves the memory consumption by using a bit representation that is useful for identifying data-independent subproblems with $O(2^n)$ space. Furthermore, the proposed method realizes further acceleration with MITM.

As for approximation approaches, these approaches return a solution that is theoretically guaranteed to be close to the optimal solution. Sahni and Gonzalez [10] proved that there is no algorithm that solves the sTSP in polynomial time. However, a polynomial time algorithm exists for the sTSP if edge weights in the graph satisfy the triangle inequality. Christofides [4] presented a 1.5 approximation algorithm for the sTSP. The worst time complexity of this algorithm is $O(n^3)$, which is better than that of the Held-Karp algorithm. Therefore, approximation approaches are useful when the overlying application accepts solutions being within a certain bound.

In contrast to the approaches mentioned above, heuristic approaches are usually based on a rule of thumb, which returns a practical solution but not guaranteed to be optimal or within certain error bounds. The Lin-Kernighan algorithm [5] deploys an iterative approach that improves an initial solution by removing two or more edges and adding the same number of edges. The error to the optimal solution was usually 1–2%, making heuristic approaches more attractive than approximation approaches [7]. However, Chandra *et al.* [11] pointed out that the computed weight could be $4\sqrt{n}$ times larger than the optimal value. Consequently, approximation approaches must be selected if the overlying application requires guarantee on the maximum error.

3. Held-Karp Algorithm

Let $V = \{0, 1, \cdots, n-1\}$ be the set of vertices that compose a graph G. Each vertex then can be identified with an integer value ranging from 0 to $n-1$. The optimal path can be obtained by determining the best visiting order starting from a vertex and ending at the same vertex. In the following, let vertex $n-1$ be the starting/ending vertex. Accordingly, we determine the visiting order for the remaining vertices, which compose a set $V' = V \setminus \{n-1\} = \{0, 1, \cdots, n-2\}$.

To describe how the Held-Karp algorithm computes the optimal value OPT, we define the function tsp as follows.

Definition 1. *For $S \subseteq V'$ and $x \in S$, let* $\mathrm{tsp}(S, x)$ *be the minimum weight of the path that starts from the starting vertex $n-1$, visits all vertices in set S, and reaches vertex x.*

The optimal value OPT and $\mathrm{tsp}(S, x)$ then can be given by Eqs. (1) and (2), respectively [2].

$$\mathrm{OPT} = \min_{x \in V'}(\mathrm{tsp}(V', x) + w(x, n-1)), \tag{1}$$

$$\mathrm{tsp}(S, x) = \begin{cases} w(n-1, x), & \text{if } |S| = 1, \\ \min\limits_{y \in S \setminus \{x\}} (\mathrm{tsp}(S \setminus \{x\}, y) + w(y, x)), & \text{otherwise.} \end{cases} \tag{2}$$

According to Eqs. (1) and (2), each value of $\mathrm{tsp}(S, x)$, where $S \in \mathcal{P}(V') \setminus \phi$ and $x \in S$, is accessed $\max(n-1-|S|, 1)$ times to obtain OPT. $\mathcal{P}(\cdot)$ here is the power set and ϕ is the empty set. Figure 1 shows how these references occur during OPT computation, where the values of $\mathrm{tsp}(S, x)$ are accessed multiple times. This data dependence relation motivates the Held-Karp algorithm to deploy dynamic programming to avoid redundant computation; the algorithm stores the values of $\mathrm{tsp}(S, x)$ in table $\mathrm{D_{tsp}}$ to reuse the computed values without duplicated computation. This table can be indexed by pair (S, x), which have $\sum_{k=1}^{n-1} k\binom{n-1}{k}$ cells, where $\binom{n}{k}$ is the number of k-combinations from n elements, because x can take k different vertices when $|S| = k$, where $1 \leq k \leq n-1$ (Fig. 1).

Next, the optimal path relies on the following definitions.

Definition 2. *Let $R = r_1 \to r_2 \to \cdots \to r_n$ be the path that visits n vertices in the following order: $r_1, r_2, \cdots, r_n$.*

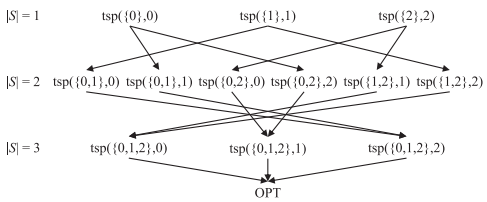

Fig. 1 Data dependence relation inherent in OPT computation for $n = 4$. The value of $\mathrm{tsp}(\{0\}, 0)$ is required to compute the values of $\mathrm{tsp}(\{0, 1\}, 1)$ and $\mathrm{tsp}(\{0, 2\}, 2)$.

Definition 3. *Consider a path that has the weight of* $\mathrm{tsp}(S, x)$*; the path starts from the starting vertex* $n-1$*, visits all vertices in set* $S \subseteq V'$*, and reaches vertex* $x \in S$*. Let* $\mathrm{prev}(S, x) \in V$ *denote the vertex visited immediately before vertex* x *on the path.*

The vertex $\mathrm{prev}(S, x)$ then can be computed according to Eq. (3) as follows.

$$\mathrm{prev}(S, x) = \begin{cases} n-1, & \text{if } |S| = 1, \\ \arg\min_{y \in S \setminus \{x\}} (\mathrm{tsp}(S \setminus \{x\}, y) + w(y, x)), & \text{otherwise}, \end{cases} \tag{3}$$

where the function $\arg\min(\cdot)$ returns an argument that minimizes the value of the expression given to the function.

According to the above definitions, the optimal path $n-1 \to v_{n-1} \to \cdots \to v_1 \to n-1$, where v_k represents the k-th vertex visited before reaching the ending vertex $n-1$, can be computed as follows.

$$v_1 = \arg\min_{x \in V'} \left(\mathrm{tsp}(V', x) + w(x, n-1)\right), \tag{4}$$

$$v_k = \begin{cases} \mathrm{prev}(V', v_1), & \text{if } k = 2, \\ \mathrm{prev}(V' \setminus \{v_1, v_2, \cdots, v_{k-2}\}, v_{k-1}), & \text{otherwise}. \end{cases} \tag{5}$$

That is, the optimal path can be identified from v_1 by Eq. (4), then $v_2, v_3, \cdots, v_{n-1}$ by Eq. (5).

Note here that the value of $\mathrm{prev}(S, x)$ can be simultaneously computed with that of $\mathrm{tsp}(S, x)$. Similar to $\mathrm{tsp}(S, x)$, which we mentioned earlier, redundant computation can be avoided by storing the value of $\mathrm{prev}(S, x)$ in table $\mathrm{D}_{\mathrm{prev}}$.

3.1 Complexity Analysis

We first analyze $\mathrm{L}(n)$, or the space complexity of the Held-Karp algorithm. We assume that a single value occupies constant space. $\mathrm{L}(n)$ then can be given by

$$\mathrm{L}(n) = \mathrm{L}_{\mathrm{tsp}}(n) + \mathrm{L}_{\mathrm{prev}}(n), \tag{6}$$

where $\mathrm{L}_{\mathrm{tsp}}(n)$ and $\mathrm{L}_{\mathrm{prev}}(n)$ are the space complexities for tables $\mathrm{D}_{\mathrm{tsp}}$ and $\mathrm{D}_{\mathrm{prev}}$, respectively. Since these tables have the same indexing scheme (S, x), where $S \in \mathcal{P}(V') \setminus \phi$ and $x \in S$, we have

$$\mathrm{L}_{\mathrm{tsp}}(n) = \mathrm{L}_{\mathrm{prev}}(n) = \sum_{k=1}^{n-1} k \binom{n-1}{k}. \tag{7}$$

Thus, the worst-case space complexity of the Held-Karp algorithm is $O(n2^n)$ because the binomial theorem rewrites Eq. (7) into $\mathrm{L}_{\mathrm{tsp}}(n) = \mathrm{L}_{\mathrm{prev}}(n) = (n-1)2^{n-2}$.

We next investigate $\mathrm{T}(n)$, or the time complexity of the Held-Karp algorithm, assuming a comparison as the elementary operation. We define subproblem $\mathrm{P}(S, x)$ as follows.

Definition 4. *For* $S \subseteq V'$ *and* $x \in S$*, let* $\mathrm{P}(S, x)$ *denote the subproblem that computes the values of* $\mathrm{tsp}(S, x)$ *and* $\mathrm{prev}(S, x)$*, i.e., fills in tables* $\mathrm{D}_{\mathrm{tsp}}$ *and* $\mathrm{D}_{\mathrm{prev}}$.

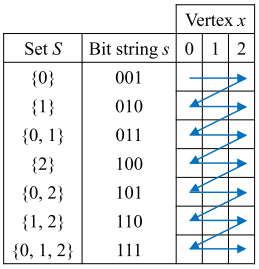

Fig. 2 Data structure of the previous method [7]. Tables are stored in two-dimensional arrays that have row s and column x. Arcs on the table represent the execution order for filling up the table.

$\mathrm{T}(n)$ can be given by

$$\mathrm{T}(n) = \mathrm{T}_1(n) + \mathrm{T}_2(n), \tag{8}$$

where $\mathrm{T}_1(n)$ is the time complexity for solving a set of all subproblems, $\{\mathrm{P}(S, x) \mid S \in \mathcal{P}(V') \setminus \phi, x \in S\}$, and $\mathrm{T}_2(n)$ is the time complexity for obtaining the optimal value and the optimal path from tables $\mathrm{D}_{\mathrm{tsp}}$ and $\mathrm{D}_{\mathrm{prev}}$, respectively. T_1 and T_2 can be expressed as follows.

$$\mathrm{T}_1(n) = \sum_{k=2}^{n-1} k(k-1) \binom{n-1}{k}, \tag{9}$$

$$\mathrm{T}_2(n) = n-1. \tag{10}$$

Thus, the worst-case time complexity of the Held-Karp algorithm is $O(n^2 2^n)$ because the binomial theorem rewrites Eq. (9) into $\mathrm{T}_1(n) = (n-1)(n-2)2^{n-3}$.

3.2 Previous Method

The previous method [7], which we denote hereafter by M_0, deploys a bit string expression to represent set $S \subseteq V'$ (Fig. 2). That is, set S is expressed by bit string s, or a binary sequence of $n-1$ bits; the x-th bit of s, where $0 \le x \le n-2$, is 1 if $x \in S$ and 0 otherwise. Consequently, bit string s can be interpreted as an integer value in the range $[0, 2^{n-1} - 1]$. This value can be directly used as an index of tables $\mathrm{D}_{\mathrm{tsp}}$ and $\mathrm{D}_{\mathrm{prev}}$, which can be stored in two-dimensional arrays. As shown in Fig. 2, the tables are indexed with row s and column x to store the value for pair (S, x), where $S \in \mathcal{P}(V') \setminus \phi$ and $x \in S$. Thus, table values can be rapidly accessed in constant time.

This indexing scheme has another benefit in obtaining a correct execution order for solving subproblems. As shown in Fig. 1, some subproblems must be computed before others due to data dependence. Filling up the arrays from lower to higher indices guarantees such a correct execution order (Fig. 2).

4. Proposed Method

The Held-Karp algorithm produces not only data-dependent

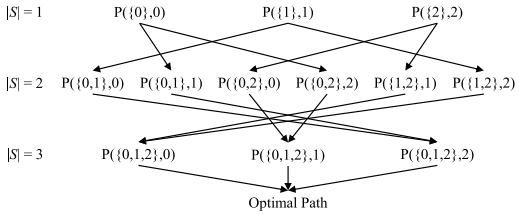

Fig. 3 An example of data-independent subproblems that can be processed in parallel ($n = 4$). Subproblems $\{P(S, x)\}$ of the same size $|S|$ can be processed in parallel, but those with different $|S|$ must be processed sequentially (from the top to bottom of the figure).

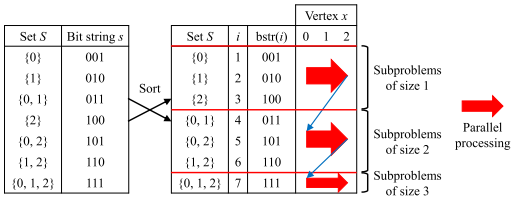

Fig. 4 Proposed data structure that facilitates locating data-independent subproblems ($n = 4$). Subproblems are sorted by Hamming weight to store data-independent subproblems in a contiguous region. Note here that we omit the 0-th row ($i = 0$) in this figure, because the 0-th row corresponds to the subproblem of size zero (i.e., an empty set). However, this dummy row is required to simplify index computation.

subproblems but also data-independent subproblems. The proposed method M_1 identifies such data-independent subproblems to process them in parallel. The method M_2 further achieves acceleration by MITM, which exploits the symmetric attribute available in the graph. The method M_2 also reduces the memory usage.

4.1 Parallelization Technique

The proposed method M_1 classifies subproblems into $n - 1$ groups such that subproblems in each group can be processed in parallel. In other words, subproblems belonging to different groups must be processed sequentially with $n-1$ steps. The key idea for realizing this classification is to group subproblems by their size k, where $1 \le k \le n - 1$.

Definition 5. *Let* $|P(S, x)| = |S|$ *be the size of subproblem* $P(S, x)$.

More specifically, Eqs. (2) and (3) indicate that subproblems of size k depend only on those of size $k-1$ (Fig. 3). This implies that (1) at least $n - 1$ steps are required to process all $1 \le k \le n-1$ and (2) subproblems of the same size k are data-independent, allowing them to be solved in parallel.

Because the proposed method adopts the same indexing scheme as the previous method M_0 [7], the subproblem size $|S|$ can be easily computed according to the Hamming weight of bit string s, where s corresponds to set $S \subseteq V'$.

Definition 6. *Let* $d(s)$ *be the number of 1s in bit string* s.

Thus, we have $|S| = d(s)$ for any S, which means that data-independent subproblems can be classified into the same group by Hamming weight.

Notice here that (data-independent) subproblems that have the same Hamming weight involve non-contiguous rows in each table because the previous method M_0 stores subproblems in an increasing order of s (Fig. 2). For example, subproblems of size 1 exist in the first, second, and fourth rows in Fig. 2. Thus, search operations are required to locate subproblems that form the same group, eliminating the advantage of constant-time access. To deal with this issue, our method sorts the subproblems, i.e., bit strings, by Hamming weight so that data-independent subproblems are stored in contiguous rows in each table (Fig. 4). This operation incurs an overhead, but once sorted, the data-independent subproblems can be located in constant time.

We introduce functions bstr and offset to identify the contiguous rows that correspond to data-independent subproblems.

Definition 7. *Let* $\mathrm{bstr}(i)$*, where* $0 \le i \le 2^{n-1} - 1$*, be the* i*-th bit string in a binary sequence of length* $n - 1$ *sorted by ascending order of Hamming weight where bit sequences of the same Hamming weight are further sorted by ascending order of numerical value.*

Definition 8. *Let* $\mathrm{offset}(k)$ *be the number of bit strings with Hamming weight at most* $k - 1$ *given by*

$$\mathrm{offset}(k) = \sum_{l=0}^{k-1} \binom{n-1}{l}, \tag{11}$$

where $1 \le k \le n - 1$.

The number $\mathrm{offset}(k)$ implies the minimum index for bit sequences of hamming weight of k. Consequently, sorted subproblems of size k exist in a contiguous region from bit strings $\mathrm{bstr}(\mathrm{offset}(k))$ to $\mathrm{bstr}(\mathrm{offset}(k + 1) - 1)$. To locate these rows rapidly, we precompute the function values of $\mathrm{bstr}(i)$ and $\mathrm{offset}(k)$ for all i and k, which are then stored in tables D_{bstr} and D_{offset}, respectively.

We first analyze $L'(n)$, or the space complexity of M_1, which can be expressed as follows.

$$L'(n) = L_{\mathrm{bstr}}(n) + L_{\mathrm{offset}}(n) + L_{\mathrm{tsp}}(n) + L_{\mathrm{prev}}(n), \tag{12}$$

where $L_{\mathrm{bstr}}(n)$ and $L_{\mathrm{offset}}(n)$ are the space complexities for storing the values of $\mathrm{bstr}(i)$ and $\mathrm{offset}(k)$, for all $0 \le i \le 2^{n-1} - 1$ and $1 \le k \le n - 1$, respectively. Obviously, we have $L_{\mathrm{bstr}} \in O(2^n)$ and $L_{\mathrm{offset}} \in O(n)$. Thus, L_{bstr} and L_{offset} are asymptotically negligible in comparison to $L_{\mathrm{tsp}}, L_{\mathrm{prev}} \in O(n2^n)$. Therefore, the space complexity of M_1 equals to that of M_0.

Finally, $T'(n)$, or the time complexity of M_1, can be given by

$$T'(n) = T_{\mathrm{bstr}}(n) + T_{\mathrm{offset}}(n) + T_1(n) + T_2(n), \tag{13}$$

where $T_{\mathrm{bstr}}(n)$ and $T_{\mathrm{offset}}(n)$ are the time complexities for computing the values of $\mathrm{bstr}(i)$ and $\mathrm{offset}(k)$, for all $0 \le i \le 2^{n-1} - 1$ and $1 \le k \le n - 1$, respectively.

- $T_{\mathrm{bstr}}(n) \in O(n2^n)$ holds. To compute the values of

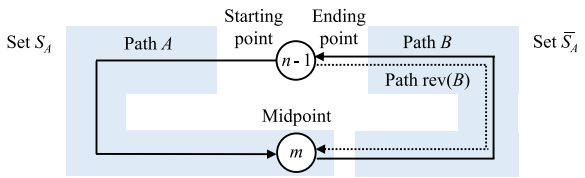

Fig. 5 MITM technique for computing the optimal path from both clockwise and counterclockwise directions.

bstr(i), we have to obtain the Hamming weights of all bit strings. Since the number of bit strings of length $n-1$ is 2^{n-1}, the worst-case time complexity for obtaining all Hamming weights is given by $O(n2^n)$. As for sorting subproblems, the merge sort algorithm has the worst-case time complexity of $O(n2^n)$.
- $\mathrm{T}_{\mathrm{offset}}(n) \in O(n2^n)$ holds. The values of offset(k) can be computed by traversing row indices from the top of table $\mathrm{D}_{\mathrm{bstr}}$; offset($k$) is the first index that has Hamming weight k.

Thus, the terms in Eq. (13) except T_1 are asymptotically negligible in comparison to T_1. Since T_1 is the time complexity for solving all subproblems, parallelizing this performance bottleneck leads to acceleration.

4.2 Meet in the Middle (MITM) Technique

The proposed method M_2 integrates the MITM technique into M_1. Given an undirected graph, MITM computes the optimal path from both clockwise and counterclockwise directions, solving subproblems of at most size $\lceil n/2 \rceil$ instead of those of at most $n-1$ (Fig. 5). In comparison to methods M_0 and M_1, MITM roughly halves the number of subproblems that are to be solved.

We first explain how MITM obtains the optimal value. To do this, we introduce the following operators for paths.

Definition 9. *Let + be the binary operator that concatenates two paths. That is, we obtain path* $R + Q = r \to \cdots \to x \to \cdots \to q$ *from paths* $R = r \to \cdots \to x$ *and* $Q = x \to \cdots \to q$.

Definition 10. *Let* rev be the unary operator that returns the given path in reverse order. That is, we obtain path $\mathrm{rev}(R) = r_n \to r_{n-1} \to \cdots \to r_1$ from path $R = r_1 \to r_2 \to \cdots \to r_n$.

Theorem 1. *Obviously,* $\mathrm{rev}(\mathrm{rev}(R)) = R$ *holds.*

Let m be the midpoint of the optimal path (Fig. 5); since the optimal path contains n vertices, m is the $\lceil n/2 \rceil$-th vertex from the starting vertex. Consequently, the optimal path can be given by $A+B$, where $A = n-1 \to a_1 \to \cdots \to a_{\lceil n/2 \rceil - 1} \to m$ and $B = m \to b_1 \to \cdots \to b_{n-\lceil n/2 \rceil - 1} \to n-1$. Theorem 2 holds for the weights of paths A and B.

Theorem 2. *Let* S_A *be the set of vertices that constitute path* A. *The weights of paths* A *and* B *are then given by* $\mathrm{tsp}(S_A, m)$ *and* $\mathrm{tsp}(\overline{S_A} \cup \{m\}, m)$, *respectively, where* $\overline{S_A}$ *is the complementary set of* S_A *in* V'.

Proof. According to Definition 1, the weights of paths A and $\mathrm{rev}(B)$ are $\mathrm{tsp}(S_A, m)$ and $\mathrm{tsp}(\overline{S_A} \cup \{m\}, m)$, respectively. Since the graph consists of undirected edges, i.e., weights are symmetric, the weight of path B equals to that of path $\mathrm{rev}(B)$. Therefore, the weight of path B is given by $\mathrm{tsp}(\overline{S_A} \cup \{m\}, m)$. □

Algorithm 1 Optimal path computation with MITM.
1: Compute set S_A and midpoint m by Eq. (16);
2: Compute path A by Eq. (5), which starts from $\mathrm{prev}(S_A, m)$;
3: Compute path $\mathrm{rev}(B)$ by Eq. (5), which starts from $\mathrm{prev}(\overline{S_A} \cup \{m\}, m)$;
4: Compute the optimal path $A + \mathrm{rev}(\mathrm{rev}(B))$;

Equation (14) gives the optimal value OPT.

$$\mathrm{OPT} = \min_{(S,x) \in F} \Big(\mathrm{tsp}(S, x) + \mathrm{tsp}(\overline{S} \cup \{x\}, x) \Big), \tag{14}$$

where set F is given by

$$F = \{(S, x) \mid S \in \mathcal{P}(V') \setminus \phi, x \in S \text{ such that } |S| = \lceil n/2 \rceil\}. \tag{15}$$

Once we set a pair (S, x) for the first term of Eq. (14), we accordingly obtain a unique pair $(\overline{S} \cup \{x\}, x)$ for the second term. Furthermore, $|\overline{S} \cup \{x\}| \le \lceil n/2 \rceil$ holds. Therefore, we can compute OPT from subproblems of at most size $\lceil n/2 \rceil$.

We next describe how the proposed method M_2 obtains the optimal path with MITM. Algorithm 1 returns the optimal path from set S_A and midpoint m, which can be given by

$$(S_A, m) = \arg\min_{(S,x) \in F} \Big(\mathrm{tsp}(S, x) + \mathrm{tsp}(\overline{S} \cup \{x\}, x) \Big). \tag{16}$$

We next analyze $\mathrm{T}''(n)$, or the time complexity of M_2, which is given by

$$\mathrm{T}''(n) = \mathrm{T}_{\mathrm{bstr}}(n) + \mathrm{T}_{\mathrm{offset}}(n) + \mathrm{T}'_1(n) + \mathrm{T}'_2(n), \tag{17}$$

where $\mathrm{T}'_1(n)$ is the time complexity for solving subproblems of size $k \le \lceil n/2 \rceil$ and $\mathrm{T}'_2(n)$ is that for obtaining the optimal value and path. The former term is given by

$$\mathrm{T}'_1(n) = \sum_{k=2}^{\lceil n/2 \rceil} k(k-1) \binom{n-1}{k}. \tag{18}$$

Lemmas 1 and 2 then give Theorem 3 whose proofs are presented in Appendix A, Appendix B and Appendix C, respectively.

Lemma 1. *The time complexity* $\mathrm{T}'_1(n)$ *for solving all subproblems of size* $k \le \lceil n/2 \rceil$ *is given by*

$$\mathrm{T}'_1(n) = \begin{cases} \mathrm{T}_1(n)/2, & \text{if } n \text{ is even,} \\ \mathrm{T}_1(n)/2 + \dfrac{(n-1)(n-2)}{2} \dbinom{n-3}{(n-3)/2}, & \text{otherwise.} \end{cases} \tag{19}$$

Lemma 2. *For any even number n,* $\binom{n}{n/2} \in \Theta(2^n/\sqrt{n})$ *holds.*

Theorem 3. *Given the time complexity* $\mathrm{T}_1(n)$ *for solving subproblems for all* $1 \le k \le n-1$ *and that* $\mathrm{T}'_1(n)$ *for solving subproblems for all* $k \le \lceil n/2 \rceil$*, we have*

$$\lim_{n\to\infty} \mathrm{T}'_1(n)/\mathrm{T}_1(n) = 1/2. \tag{20}$$

As for the latter term of Eq. (17), we have

$$\mathrm{T}'_2(n) = \lceil n/2 \rceil \binom{n-1}{\lceil n/2 \rceil} - 1, \tag{21}$$

because there are $\binom{n-1}{\lceil n/2 \rceil}$ sets whose size is $|S| = \lceil n/2 \rceil$. According to Lemma 2, $\mathrm{T}'_2(n) \in O(\sqrt{n}2^n)$ holds. Therefore, similar to $\mathrm{T}_{\mathrm{bstr}}(n)$ and $\mathrm{T}_{\mathrm{offset}}(n)$, $\mathrm{T}'_2(n)$ is asymptotically negligible in comparison to T_1 in Eq. (17). In summary, MITM is expected to double the performance according to Theorem 3.

Note that MITM can be integrated into the previous method M_0 instead of M_1; this integration allows us to skip subproblems of larger than $\lceil n/2 \rceil$. However, we avoid this because MITM is able to reduce the memory usage if subproblems in tables $\mathrm{D}_{\mathrm{tsp}}$ and $\mathrm{D}_{\mathrm{prev}}$ are sorted by Hamming weight. Thus, there exists a common procedure in methods M_1 and M_2.

4.3 Reducing Memory Usage

The proposed method M_2 is allowed to store the solutions for the subproblems of size $k \le \lceil n/2 \rceil$ whereas the previous method M_0 has to store the solutions for those of size $k \le n-1$. Therefore, the memory usage of M_2 asymptotically reduces to $1/2$ in comparison to that of M_0 as follows.

Theorem 4. *Let* $\mathrm{L}'_{\mathrm{tsp}}(n)$ *and* $\mathrm{L}'_{\mathrm{prev}}(n)$ *be the space complexities of tables* $\mathrm{D}_{\mathrm{tsp}}$ *and* $\mathrm{D}_{\mathrm{prev}}$ *in method* M_2*, respectively. We then have*

$$\lim_{n\to\infty} \mathrm{L}'_{\mathrm{tsp}}(n)/\mathrm{L}_{\mathrm{tsp}}(n) = 1/2, \tag{22}$$

$$\lim_{n\to\infty} \mathrm{L}'_{\mathrm{prev}}(n)/\mathrm{L}_{\mathrm{prev}}(n) = 1/2. \tag{23}$$

The proof for Theorem 4 is presented in Appendix D.

The above discussion assumes that necessary rows in tables $\mathrm{D}_{\mathrm{tsp}}$ and $\mathrm{D}_{\mathrm{prev}}$ exist in contiguous regions as shown in Fig. 6. In other words, tables in the previous method M_0 fails to save the memory usage because necessary rows exist in non-contiguous regions; the entire table must be allocated for execution.

Similar to the original method M_0, the tables must be accessed in constant time to achieve acceleration with MITM. To realize this constant time access, we introduce function rank as follows.

Definition 11. *Consider the i-th bit string s, where* $0 \le i \le 2^{n-1}-1$ *and* $s = \mathrm{bstr}(i)$*, in a sequence of bit strings sorted by ascending order of Hamming weight where bit sequences of the same Hamming weight are further sorted by ascending order of numerical value. Let* $\mathrm{rank}(s)$ *be the rank of string s in the sequence of bit strings that have the same Hamming weight.* $\mathrm{rank}(s)$ *is then given by* $\mathrm{rank}(s) = i - \mathrm{offset}(\mathrm{d}(s))$.

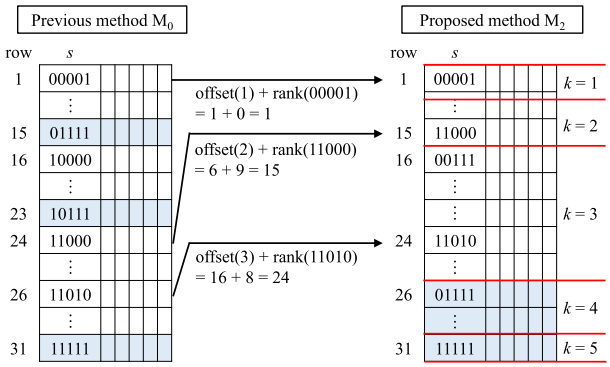

Fig. 6 Comparison of data structures deployed for table $\mathrm{D}_{\mathrm{prev}}$ of the previous method M_0 and that of the proposed method M_2 ($n = 6$). Method M_2 stores necessary rows in a contiguous region, allowing unnecessary rows not to be allocated. Unnecessary rows are colored with light blue.

Using the function rank, the value for (s, x) at the k-th step can be stored in the element at $(\mathrm{offset}(k) + \mathrm{rank}(s), x)$ of the two-dimensional array. As shown in Fig. 6, this indexing scheme allows necessary rows to be stored in a contiguous region and to be accessed in constant time. For example, a bit sequence of 11010 has a hamming weight of 3. Accordingly, this sequence corresponds to the subproblems of size 3, which are processed at the third step ($k = 3$). We then have $\mathrm{offset}(3) = 16$ according to Definition 8. Furthermore, we have $\mathrm{rank}(11010) = 8$ because $\mathrm{rank}(11010)$ gives the rank of 11010 in sorted sequences that have a hamming weight of 3: 00111, 01011, 01101, 01110, 10011, 10101, 10110, 11001, 11010 and 11100, each corresponding to 7, 11, 13, 14, 19, 21, 22, 25, 26 and 28 in decimal.

The time complexity $\mathrm{T}_{\mathrm{rank}}(n)$ for computing $\mathrm{rank}(s)$ is negligible in comparison to T'_1. In more detail, $\mathrm{T}_{\mathrm{rank}}(n) \in O(2^n)$ holds because the values of $\mathrm{rank}(s)$ can be computed by traversing row indices from the top of table $\mathrm{D}_{\mathrm{bstr}}$. Similarly, the space complexity $\mathrm{L}_{\mathrm{rank}}(n)$ for storing $\mathrm{rank}(s)$ is negligible in comparison to $\mathrm{L}_{\mathrm{tsp}}$ and $\mathrm{L}_{\mathrm{prev}}$; $\mathrm{L}_{\mathrm{rank}}(n) \in O(2^n)$ holds if we store all values in table $\mathrm{D}_{\mathrm{rank}}$.

The space complexity for table $\mathrm{D}_{\mathrm{tsp}}$ can be further reduced by allocating rows only for two consecutive steps and overwriting the allocated rows that are not to be accessed. More specifically, $\mathrm{tsp}(S, x)$ at the $(k+1)$-th step depends only on that at the k-th step, which means that all values of $\mathrm{tsp}(S, x)$ except those at the last step can be overwritten. Accordingly, the proposed method M_2 allocates only the rows for the latest two steps rather than all rows in table $\mathrm{D}_{\mathrm{tsp}}$. Furthermore, the method deploys a double buffering scheme that stores the rows computed at odd and even steps in tables $\mathrm{D}_{\mathrm{tsp1}}$ and $\mathrm{D}_{\mathrm{tsp0}}$, respectively (Fig. 7). Each table holds the value for bit string s in the $\mathrm{rank}(s)$-th row to realize constant time access to a contiguous region.

Finally, we analyze the space complexities of tables $\mathrm{D}_{\mathrm{tsp0}}$ and $\mathrm{D}_{\mathrm{tsp1}}$. The number of values computed at the k-th step is maximized when $k = \lceil (n-1)/2 \rceil$. Each table

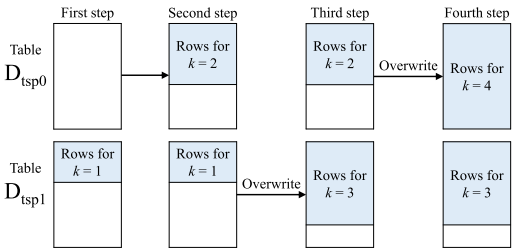

Fig. 7 Double buffering scheme that alternatively uses tables D_{tsp0} and D_{tsp1} ($n = 8$). Instead of allocating all rows in D_{tsp}, limited rows are allocated only for two consecutive steps.

then requires $\binom{n-1}{\lceil (n-1)/2 \rceil}$ rows and $n-1$ columns to store all values generated at the k-th step. According to Lemma 2, both the space complexities for tables D_{tsp0} and D_{tsp1} can be given by $O(\sqrt{n}2^n)$, which is negligible in comparison to $L_{prev} \in O(n2^n)$.

5. Experiments

We evaluated the proposed methods M_1 and M_2 by comparing them with the previous method M_0. The evaluation metrics were the execution time, the speedup ratio, and the memory usage, which demonstrated the impact of parallelization and MITM.

For the experiments, all methods were implemented with the C language and OpenMP directives [12] to obtain performance results on a multi-core CPU. We adopted bucket sort to implement sorting operations on the CPU. Similarly, we implemented all methods except M_0 with the compute unified device architecture (CUDA) [13] to execute the methods on a GPU. Sort operations were implemented with the Thrust library [14]. Table 1 shows the specification of the deployed machine. Hyper-threading was enabled on the 8-core CPU, and thus, at most 16 threads were executed on 16 logical cores. Our CPU-based implementation parallelizes the k-th step with all CPU threads. On the other hand, the GPU-based implementation parallelizes the k-th step with $\binom{n-1}{k}$ threads; each thread is responsible for processing a bit sequence and each block consists of 1024 threads.

As for the problem size n, we varied the number of vertices from 4 to 29, which was the maximum number that could be solved on the GPU successfully; the execution for $n \geq 30$ failed due to exhaustion of GPU memory. Given a complete weighted graph, all the methods returned the optimal path with its value OPT. Each edge in the graph had a random weight specified as an integer value in the range $[1, 1023]$. For each size of n, we generated 10 graphs that had random weights. For each graph, we executed 50 times (500 times for each n) and the minimum time was selected for each n. We found that the maximum time was close to the minimum time. For example, the execution times of M_2 on the GPU ranged from 1.16 s to 1.17 s for $n = 29$. Thus, the gap between the minimum and maximum times was negligible on our machine because the time complexity of the Held-Karp algorithm depends only on n.

Table 1 Specification of experimental machine.

Classification	Specification
CPU	Intel Xeon Silver 4110 (8 cores)
Memory capacity (CPU)	96 GB
GPU	NVIDIA Tesla V100 PCIe
Memory capacity (GPU)	16 GB
GPU driver	396.44
OS	Ubuntu 16.045 LTS
Development environment	GCC 5.4.0 and CUDA 9.2
Compile option	03 optimization

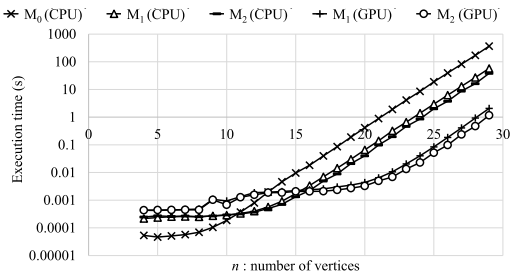

Fig. 8 Comparison of the previous method M_0 and the proposed methods M_1 and M_2 in terms of execution time.

Figure 8 shows the execution time for different problem sizes ($4 \leq n \leq 29$). The proposed methods M_1 and M_2 were faster than the previous method M_0 when the problem size was sufficiently large ($n \geq 13$) where efficient parallelization was achieved. In other words, small problem sizes solved within 0.001 s could not be accelerated by parallelization, which involved a synchronization overhead; a synchronization operation is necessary at every step to increase the problem size k. We also found that M_2 was faster than M_1 because acceleration by MITM is independent from that by parallelization. That is, the former technique reduces the number of subproblems that are to be solved whereas the latter technique reduces the execution time for solving subproblems. In particular, M_2 running on the GPU was 30–40 times faster than that on the CPU.

All of execution times in Fig. 8 include data transfer time. The data transfer time of M_2 on the GPU for $n = 29$ was 65.2 μs, which was negligible compared to the execution time of 1.16 s. Note that data transfer occurs only at the program initialization and finalization to send the input to the GPU and receive the output from the GPU, respectively. The input size is given by $O(n^2)$, which corresponds to an adjacency matrix that stores graph information. On the other hand, the output size is given by $O(n)$, which corresponds to the optimal path and value. Thus, the amount of data transfer is asymptotically negligible to that of computation, which has the time complexity of $O(n^2 2^n)$. Note that our GPU-based implementation failed to overcome the CPU-based implementation when $n < 16$, because the problem size was not sufficient large to maximize the efficiency of GPU execution. The data transfer time for $n = 4$ occupied 12.6 % of the execution time.

Figure 9 shows the speedup ratio obtained on the CPU.

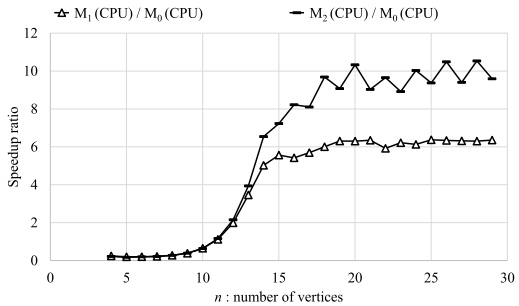

Fig. 9 Speedup ratio of the proposed methods M_1 and M_2 over the previous method M_0 with different problem sizes. All methods were executed on the CPU.

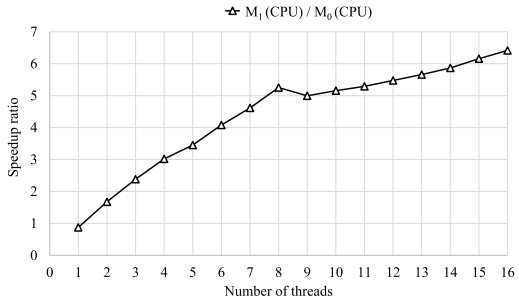

Fig. 10 Speedup ratio of the proposed method M_1 over the previous method M_0 with different numbers of CPU threads ($n = 29$). Executions using more than 8 threads involved multiple threads to be executed on a physical CPU core.

Our parallelization technique M_1 achieved a 6.3× speedup for large problems of $n \geq 25$. More speedups were achieved by M_2, which integrated the MITM technique into M_1; we obtained a 9.5× speedup when n was odd and a 10.5× speedup otherwise. Thus, acceleration achieved by MITM reached a factor of 1.5 and that of 1.7 when n was odd and even, respectively. This zigzag behavior observed in Fig. 9, i.e., higher speedups for even numbers of vertices, perfectly matches to our theoretical analysis presented in Sect. 4.2.

We next investigated how our parallelization technique M_1 increased the speedup with the number of CPU threads (Fig. 10). As shown in this figure, we observed a linear speedup for $n = 29$, which increased in proportion to the number of threads. When $n = 29$, there were many subproblems sufficient to make parallel threads busy. For example, the proposed method generated $\lceil n/2 \rceil \binom{n-1}{\lceil n/2 \rceil}$ subproblems at the $\lceil n/2 \rceil$-th step, so that $561,632,400$ subproblems were assigned to 16 CPU threads when $n = 29$. In Fig. 10, the speedup slightly decreased when we increased the number of threads from 8 to 9. This slight decrease was due to hyper-threading, which dropped the efficiency of thread execution when multiple threads were assigned to a single physical core. A similar behavior can be found in other memory-intensive applications [15]–[17].

Finally, we investigated the memory usage of the proposed and previous methods. The memory usage was computed assuming that tables D_{prev}, D_{tsp}, D_{bstr}, and D_{rank} have 1 B, 2 B, 4 B, and 4 B values, respectively. Note that we

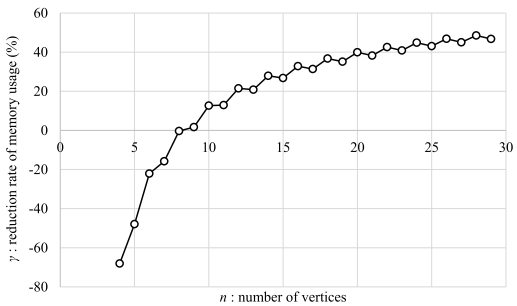

Fig. 11 Reduction rate γ of memory usage. A positive rate means that the proposed method M_2 reduced the memory usage of the previous method M_0. Execution of $n = 30$ failed due to exhaustion of GPU memory.

avoided considering variables that can be stored in polynomial space; In fact, these variables consume several KB when $n = 29$, which is negligible to other data. Figure 11 shows the reduction rate of memory usage, $\gamma = (\alpha - \beta)/\alpha$, where α and β are the memory usage of M_0 and that of M_2, respectively. We observed that the rate γ had a negative value when $n \leq 7$ but increased with n. The reason for this negative value was due to the additional tables D_{bstr} and D_{offset}. On the other hand, the increase was yielded by the memory reduction technique presented in Sect. 4.3; the proposed method replaces table D_{tsp} with two smaller tables D_{tsp0} and D_{tsp1}, which reduces the space complexity from $O(n2^n)$ to $O(\sqrt{n}2^n)$. Consequently, the gap from M_0 to M_2 increased with n.

Note here that the memory usage showed a zigzag behavior similar to that observed for the speedup (Fig. 9). This behavior was also theoretically analyzed in Sect. 4.2. The highest reduction rate of 48% was obtained when $n = 28$, which was the largest even problem size processed successfully on the deployed GPU. In theoretical, the memory usage roughly doubles as the problem size increases by 1. Actually, method M_2 consumed 5.2 GB and 11.2 GB of GPU memory when $n = 28$ and $n = 29$, respectively. Thus, the problem size of $n = 30$ required approximately 22 GB of GPU memory for execution, which was beyond the capacity available on the deployed GPU.

6. Conclusion

In this paper, we proposed an acceleration method for the Held-Karp algorithm, which is an exact algorithm for the sTSP. The proposed method achieves acceleration with parallelization and MITM. We also showed theoretical analysis for the time and space complexities of the proposed method.

Experimental results show that the proposed method on an 8-core CPU was 9.5–10.5 times faster than the previous method on a single-core CPU. Moreover, the proposed method on the latest GPU was 30–40 times faster than that on the multi-core CPU. As a side effect, the proposed method reduced the memory usage by 48%, which matches to the theoretical analysis.

Future work is to deal with large problem sizes that cannot be naively stored in the GPU memory due to memory

exhaustion. An out-of-core processing scheme [18], [19] may be useful for realizing this large-scale computation; the scheme divides data into small pieces and iteratively processes the pieces with overlapping GPU computation with CPU–GPU data transfer.

Acknowledgments

This study was supported in part by the Japan Society for the Promotion of Science KAKENHI Grant Numbers JP15H01687 and JP16H02801. We are also grateful to the anonymous reviewers for their valuable comments.

References

[1] X. Wang, B. Tang, Y. Yan, and X. Gu, "Time-optimal path planning for dual-welding robots based on intelligent optimization strategy," Trans. Intelligent Welding Manufacturing, vol.1, no.2, pp.47–59, Dec. 2017.
[2] M. Held and R.M. Karp, "A dynamic programming approach to sequencing problems," J. Society for Industrial and Applied Mathematics, vol.10, no.1, pp.196–210, March 1962.
[3] W. Cook, "Concorde home," 2016. http://www.math.uwaterloo.ca/tsp/concorde.html.
[4] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Tech. Rep., Carnegie-Mellon University Pittsburgh Pa Management Sciences Research Group, Feb. 1976.
[5] S. Lin and B.W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," Operations Research, vol.21, no.2, pp.498–516, April 1973.
[6] W.J. Cook, In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation, Princeton University Press, Dec. 2011.
[7] M. Kubo, "An invitation to the traveling salesman problem ii," Operations Research, vol.39, no.2, pp.91–96, Feb. 1994. (in Japanese).
[8] F. Ahammed and P. Moscato, "Evolving l-systems as an intelligent design approach to find classes of difficult-to-solve traveling salesman problem instances," Proc. Int'l Conf. Applications of Evolutionary Computation (EvoApplications'11), Part I, pp.1–11, April 2011.
[9] A. Moffat, "Solving the TSP with WebGL and gpgpu.js," 2017. https://amoffat.github.io/held-karp-gpu-demo/.
[10] S. Sahni and T. Gonzalez, "P-complete approximation problems," J. ACM, vol.23, no.3, pp.555–565, July 1976.
[11] B. Chandra, H. Karloff, and C. Tovey, "New results on the old k-opt algorithm for the tsp," Proc. 5th ACM-SIAM Symp. Discrete Algorithms (SODA'94), pp.150–159, Jan. 1994.
[12] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, Parallel Programming in OpenMP, Morgan Kaufmann, San Mateo, CA, Oct. 2000.
[13] NVIDIA Corporation, "CUDA C Programming Guide Version 9.2," Aug. 2018.
[14] N. Bell and J. Hoberock, Thrust: A Productivity-Oriented Library for CUDA, ch. 26, Morgan Kaufmann, San Mateo, CA, Jan. 2011. http://thrust.github.io/.
[15] Y. Matsuzaki, N. Uchikoga, M. Ohue, T. Shimoda, T. Sato, T. Ishida, and Y. Akiyama, "Megadock 3.0: a high-performance protein-protein interaction prediction software using hybrid parallel computing for petascale supercomputing environments," Source Code for Biology and Medicine, vol.8, no.1, Sept. 2013.
[16] C. Couder-Castañeda, J.C. Ortiz-Alemán, M.G.O. del Castillo, and M. Nava-Flores, "Forward modeling of gravitational fields on hybrid multi-threaded cluster," Geofísica Internacional, vol.54, no.1, pp.31–48, Jan. 2015.
[17] L.D. Antonov, C. Andreetta, and T. Hamelryck, "Parallel GPGPU evaluation of small angle X-ray scattering profiles in a markov chain monte carlo framework," Proc. 5th Int'l Joint Conf. Biomedical Engineering Systems and Technologies (BIOSTEC'12), pp.222–235, Feb. 2012.
[18] Y. Lu, F. Ino, and K. Hagihara, "Cache-aware GPU optimization for out-of-core cone beam CT reconstruction of high-resolution volumes," IEICE Trans. Inf. & Syst., vol.E99-D, no.12, pp.3060–3071, Dec. 2016.
[19] J. Shen, K. Shigeoka, F. Ino, and K. Hagihara, "GPU-based branch and bound method to solve large 0-1 knapsack problems with data-centric strategies," Concurrency and Computation: Practice and Experience, vol.31, no.4, e4954, Feb. 2019.
[20] J. Wallis, Arithmetica Infinitorum, Oxford, 1656.

Appendix A: Proof for Lemma 1

Let $\mathbb{N}_0$ be a set of non-negative integers. We introduce Lemmas 3 and 4 to show a proof for Lemma 1.

Lemma 3. *For any $n \in \mathbb{N}_0$ and $k \in \mathbb{N}_0$, where $k \le n$,*

$$k\binom{n}{k} = n\binom{n-1}{k-1}. \tag{A·1}$$

Proof. Lemma 3 obviously holds according to the definition of the binomial coefficient: $\binom{n}{k} = n!/(k!(n-k)!)$. □

Lemma 4. *For any $n \in \mathbb{N}_0$,*

$$\sum_{k=0}^{n}\binom{2n+1}{k} = 2^{2n}, \tag{A·2}$$

$$\sum_{k=0}^{n}\binom{2n}{k} = 2^{2n-1} + \frac{1}{2}\binom{2n}{n}. \tag{A·3}$$

Proof. First, we show a proof for Eq. (A·2). According to the symmetry rule for binomial coefficients, we have

$$\binom{2n+1}{k} = \binom{2n+1}{2n-k+1}, \tag{A·4}$$

for all $0 \le k \le 2n+1$. Therefore, we have

$$\sum_{k=0}^{n}\binom{2n+1}{k} = \sum_{k=n+1}^{2n+1}\binom{2n+1}{k}. \tag{A·5}$$

Using this equivalence, the left-hand side of Eq. (A·2) can be rewritten as follows.

$$\begin{aligned}\sum_{k=0}^{n}\binom{2n+1}{k} &= \frac{1}{2}\left\{\sum_{k=0}^{n}\binom{2n+1}{k} + \sum_{k=0}^{n}\binom{2n+1}{k}\right\}\\ &= \frac{1}{2}\left\{\sum_{k=0}^{n}\binom{2n+1}{k} + \sum_{k=n+1}^{2n+1}\binom{2n+1}{k}\right\}\\ &= \frac{1}{2}\sum_{k=0}^{2n+1}\binom{2n+1}{k}.\end{aligned} \tag{A·6}$$

Applying the binomial theorem to Eq. (A·6) gives

$$\sum_{k=0}^{n}\binom{2n+1}{k} = \frac{1}{2}\sum_{k=0}^{2n+1}\binom{2n+1}{k} = 2^{2n}. \tag{A·7}$$

Thus, Eq. (A· 2) holds.

We next show a proof for Eq. (A· 3). Similar to Eq. (A· 4), the symmetry rule for binomial coefficients gives

$$\binom{2n}{k} = \binom{2n}{2n-k}, \tag{A· 8}$$

for all $0 \le k \le 2n$. Therefore, we have

$$\sum_{k=0}^{n-1}\binom{2n}{k} = \sum_{k=n+1}^{2n}\binom{2n}{k}. \tag{A· 9}$$

Using this equivalence, the left-hand side of Eq. (A· 3) can be rewritten as follows.

$$\begin{aligned}\sum_{k=0}^{n}\binom{2n}{k} &= \frac{1}{2}\left\{\sum_{k=0}^{n-1}\binom{2n}{k} + \binom{2n}{n} + \sum_{k=0}^{n-1}\binom{2n}{k} + \binom{2n}{n}\right\}\\ &= \frac{1}{2}\left\{\sum_{k=0}^{n-1}\binom{2n}{k} + \binom{2n}{n} + \sum_{k=n+1}^{2n}\binom{2n}{k} + \binom{2n}{n}\right\}\\ &= \frac{1}{2}\left\{\sum_{k=0}^{2n}\binom{2n}{k} + \binom{2n}{n}\right\}. \end{aligned}\tag{A· 10}$$

Applying the binomial theorem to Eq. (A· 10) gives

$$\begin{aligned}\sum_{k=0}^{n}\binom{2n}{k} &= \frac{1}{2}\left\{\sum_{k=0}^{2n}\binom{2n}{k} + \binom{2n}{n}\right\}\\ &= 2^{2n-1} + \frac{1}{2}\binom{2n}{n}. \end{aligned}\tag{A· 11}$$

Thus, Eq. (A· 3) holds. □

Using Lemmas 3 and 4, we show a proof for Lemma 1.

Proof. By applying Lemma 3 twice to Eq. (18), we obtain

$$\begin{aligned}\mathrm{T}_1'(n) &= \sum_{k=2}^{\lceil n/2\rceil} k(k-1)\binom{n-1}{k}\\ &= (n-1)(n-2)\sum_{k=2}^{\lceil n/2\rceil}\binom{n-3}{k-2}\\ &= (n-1)(n-2)\sum_{l=0}^{\lceil n/2\rceil-2}\binom{n-3}{l}. \end{aligned}\tag{A· 12}$$

We then consider two cases.

Case 1. n **is even.** Since $n-3$ is an odd number, there is an integer m such that $2m+1 = n-3$. Therefore, $\lceil n/2\rceil - 2 = m$. Using these equations, we have

$$\sum_{l=0}^{\lceil n/2\rceil-2}\binom{n-3}{l} = \sum_{l=0}^{m}\binom{2m+1}{l}. \tag{A· 13}$$

Applying Lemma 4 to Eq. (A· 13), we obtain

$$\sum_{l=0}^{\lceil n/2\rceil-2}\binom{n-3}{l} = 2^{2m} = 2^{n-4}. \tag{A· 14}$$

Therefore,

$$\begin{aligned}\mathrm{T}_1'(n) &= (n-1)(n-2)\sum_{l=0}^{\lceil n/2\rceil-2}\binom{n-3}{l}\\ &= (n-1)(n-2)2^{n-4}\\ &= \mathrm{T}_1(n)/2. \end{aligned}\tag{A· 15}$$

Case 2. n **is odd.** Since $n-3$ is an even number, there is an integer m such that $2m = n-3$. Therefore, $\lceil n/2\rceil - 2 = m$. Using these equations, we have

$$\sum_{l=0}^{\lceil n/2\rceil-2}\binom{n-3}{l} = \sum_{l=0}^{m}\binom{2m}{l}. \tag{A· 16}$$

Applying Lemma 4 to Eq. (A· 16), we obtain

$$\begin{aligned}\sum_{l=0}^{\lceil n/2\rceil-2}\binom{n-3}{l} &= 2^{2m-1} + \frac{1}{2}\binom{2m}{m}\\ &= 2^{n-4} + \frac{1}{2}\binom{n-3}{(n-3)/2}. \end{aligned}\tag{A· 17}$$

Therefore,

$$\begin{aligned}\mathrm{T}_1'(n) &= (n-1)(n-2)\sum_{l=0}^{\lceil n/2\rceil-2}\binom{n-3}{l}\\ &= \mathrm{T}_1(n)/2 + \frac{(n-1)(n-2)}{2}\binom{n-3}{(n-3)/2}. \end{aligned}\tag{A· 18}$$

Thus, by the two cases above, Lemma 1 holds. □

Appendix B: Proof for Lemma 2

Proof. According to the Wallis product [20] for π, we have

$$\lim_{n\to\infty}\frac{2^{2n}}{\sqrt{n}\binom{2n}{n}} = \sqrt{\pi}. \tag{A· 19}$$

That is, using the (ε,δ)-definition of limit,

$$\forall\varepsilon>0,\ \exists n_0\in\mathbb{N}_0 \text{ s.t. } \forall n\in\mathbb{N}_0,$$
$$n>n_0 \ \Rightarrow\ \left|\frac{2^{2n}}{\sqrt{n}\binom{2n}{n}} - \sqrt{\pi}\right| < \varepsilon, \tag{A· 20}$$

which can be rewritten as

$$\exists\varepsilon_0 \text{ s.t. } 0<\varepsilon_0<\sqrt{\pi},\ \exists n_0\in\mathbb{N}_0 \text{ s.t. } \forall n\in\mathbb{N}_0,$$
$$n>n_0 \ \Rightarrow\ \left|\frac{2^{2n}}{\sqrt{n}\binom{2n}{n}} - \sqrt{\pi}\right| < \varepsilon_0. \tag{A· 21}$$

Then

$$\left|\frac{2^{2n}}{\sqrt{n}\binom{2n}{n}} - \sqrt{\pi}\right| < \varepsilon_0$$
$$\Rightarrow -\varepsilon_0 < \frac{2^{2n}}{\sqrt{n}\binom{2n}{n}} - \sqrt{\pi} < \varepsilon_0$$
$$\Rightarrow \sqrt{\pi} - \varepsilon_0 < \frac{2^{2n}}{\sqrt{n}\binom{2n}{n}} < \sqrt{\pi} + \varepsilon_0. \quad (A\cdot 22)$$

Since $0 < \varepsilon_0 < \sqrt{\pi}$ gives $\sqrt{\pi} - \varepsilon_0 > 0$,

$$\frac{1}{\sqrt{\pi}+\varepsilon_0} < \frac{\sqrt{n}\binom{2n}{n}}{2^{2n}} < \frac{1}{\sqrt{\pi}-\varepsilon_0}$$
$$\Rightarrow \frac{1}{\sqrt{\pi}+\varepsilon_0}\frac{2^{2n}}{\sqrt{n}} < \binom{2n}{n} < \frac{1}{\sqrt{\pi}-\varepsilon_0}\frac{2^{2n}}{\sqrt{n}}. \quad (A\cdot 23)$$

Using $c_1 = 1/(\sqrt{\pi}+\varepsilon_0)$ and $c_2 = 1/(\sqrt{\pi}-\varepsilon_0)$,

$$\exists n_0 \in \mathbb{N}_0 \text{ s.t. } \forall n \in \mathbb{N}_0,$$
$$n > n_0 \;\Rightarrow\; c_1\frac{2^{2n}}{\sqrt{n}} < \binom{2n}{n} < c_2\frac{2^{2n}}{\sqrt{n}}. \quad (A\cdot 24)$$

Thus, $\binom{2n}{n} \in \Theta\left(\frac{2^{2n}}{\sqrt{n}}\right)$. Assuming n be an even number, we obtain

$$\binom{n}{n/2} \in \Theta\left(\frac{2^n}{\sqrt{n}}\right). \quad (A\cdot 25)$$

□

Appendix C: Proof of Theorem 3

Proof. We consider two cases.

Case 1. n **is even.** According to Lemma 1, Eq. (20) obviously holds.

Case 2. n **is odd.** According to Lemma 1,

$$T_1'(n)/T_1(n) = 1/2 + \frac{\frac{(n-1)(n-2)}{2}\binom{n-3}{(n-3)/2}}{T_1(n)}. \quad (A\cdot 26)$$

According to Lemma 2, given an even number $n-3$, we have

$$\binom{n-3}{(n-3)/2} \in \Theta\left(\frac{2^n}{\sqrt{n}}\right). \quad (A\cdot 27)$$

Thus,

$$\frac{(n-1)(n-2)}{2}\binom{n-3}{(n-3)/2} \in \Theta(n^{1.5}2^n). \quad (A\cdot 28)$$

Since $T_1(n) = (n-1)(n-2)2^{n-3} \in \Theta(n^2 2^n)$, we obtain

$$\lim_{n\to\infty} T_1'(n)/T_1(n) = 1/2. \quad (A\cdot 29)$$

Thus, by the two cases above, Theorem 3 holds. □

Appendix D: Proof of Theorem 4

Proof. We show a proof for Eq. (22). As for Eq. (23), we omit the proof but it can be presented in a similar manner.

First, $L'_{tsp}(n)$ can be expressed as follows.

$$L'_{tsp}(n) = \sum_{k=1}^{\lceil n/2\rceil} k\binom{n-1}{k}. \quad (A\cdot 30)$$

By applying Lemma 3 to Eq. (A·30), we obtain

$$L'_{tsp}(n) = (n-1)\sum_{k=1}^{\lceil n/2\rceil}\binom{n-2}{k-1}$$
$$= (n-1)\sum_{l=0}^{\lceil n/2\rceil-1}\binom{n-2}{l}. \quad (A\cdot 31)$$

We then consider two cases.

Case 1. n **is even.** Since $n-2$ is an even number, there is an integer m such that $2m = n-2$. Therefore, $\lceil n/2\rceil - 1 = m$. Using these equations, we have

$$\sum_{l=0}^{\lceil n/2\rceil-1}\binom{n-2}{l} = \sum_{l=0}^{m}\binom{2m}{l}. \quad (A\cdot 32)$$

Applying Lemma 4 to Eq. (A·32), we obtain

$$\sum_{l=0}^{\lceil n/2\rceil-1}\binom{n-2}{l} = 2^{2m-1} + \frac{1}{2}\binom{2m}{m}$$
$$= 2^{n-3} + \frac{1}{2}\binom{n-2}{(n-2)/2}. \quad (A\cdot 33)$$

Therefore,

$$L'_{tsp}(n) = (n-1)\sum_{l=0}^{\lceil n/2\rceil-1}\binom{n-2}{l}$$
$$= (n-1)2^{n-3} + \frac{n-1}{2}\binom{n-2}{(n-2)/2}$$
$$= L_{tsp}(n)/2 + \frac{n-1}{2}\binom{n-2}{(n-2)/2}. \quad (A\cdot 34)$$

According to Lemma 2, given an even number $n-2$, we have

$$\binom{n-2}{(n-2)/2} \in \Theta\left(\frac{2^n}{\sqrt{n}}\right). \quad (A\cdot 35)$$

Thus,

$$\frac{n-1}{2}\binom{n-2}{(n-2)/2} \in \Theta(\sqrt{n}2^n). \quad (A\cdot 36)$$

Since $L_{tsp}(n) = (n-1)2^{n-2} \in \Theta(n2^n)$, we obtain

$$\lim_{n\to\infty} L'_{tsp}(n)/L_{tsp}(n) = \lim_{n\to\infty}\left\{1/2 + \frac{\frac{n-1}{2}\binom{n-2}{(n-2)/2}}{L_{tsp}(n)}\right\} = 1/2. \quad (A\cdot 37)$$

Case 2. n **is odd.** Since $n-2$ is an odd number, there is an integer m such that $2m+1 = n-2$. Therefore, $\lceil n/2\rceil - 1 = m+1$. Using these equations, we have

$$\sum_{l=0}^{\lceil n/2\rceil-1}\binom{n-2}{l} = \sum_{l=0}^{m+1}\binom{2m+1}{l} = \sum_{l=0}^{m}\binom{2m+1}{l} + \binom{2m+1}{m+1}. \quad (A\cdot 38)$$

Applying Lemma 4 to Eq. (A· 38), we obtain

$$\sum_{l=0}^{\lceil n/2\rceil-1}\binom{n-2}{l} = 2^{2m} + \binom{2m+1}{m+1} = 2^{n-3} + \binom{n-2}{(n-1)/2}. \quad (A\cdot 39)$$

According to Lemma 3,

$$\binom{n-2}{(n-1)/2} = \frac{2(n-2)}{n-1}\binom{n-3}{(n-3)/2}. \quad (A\cdot 40)$$

Therefore,

$$\begin{aligned} L'_{tsp}(n) &= (n-1)\sum_{l=0}^{\lceil n/2\rceil-1}\binom{n-2}{l} \\ &= (n-1)\left\{2^{n-3} + \frac{2(n-2)}{n-1}\binom{n-3}{(n-3)/2}\right\} \\ &= (n-1)2^{n-3} + 2(n-2)\binom{n-3}{(n-3)/2} \\ &= L_{tsp}(n)/2 + 2(n-2)\binom{n-3}{(n-3)/2}. \end{aligned} \quad (A\cdot 41)$$

According to Lemma 2, given an even number $n-3$, we have

$$\binom{n-3}{(n-3)/2} \in \Theta\left(\frac{2^n}{\sqrt{n}}\right). \quad (A\cdot 42)$$

Thus,

$$2(n-2)\binom{n-3}{(n-3)/2} \in \Theta(\sqrt{n}2^n). \quad (A\cdot 43)$$

Since $L_{tsp}(n) = (n-1)2^{n-2} \in \Theta(n2^n)$, we obtain

$$\lim_{n\to\infty} L'_{tsp}(n)/L_{tsp}(n) = \lim_{n\to\infty}\left\{1/2 + \frac{2(n-2)\binom{n-3}{(n-3)/2}}{L_{tsp}(n)}\right\} = 1/2. \quad (A\cdot 44)$$

Thus, by the two cases above, Lemma 4 holds. □

Kazuro Kimura received the B.E. degree information and computer sciences from Osaka University, Osaka, Japan, in 2019. He is currently an engineer at Forever Co., Ltd. His current research interests include computer graphics and high performance computing.

Shinya Higa received the B.E. degree information and computer sciences from Osaka University, Osaka, Japan, in 2019. He is currently working toward the M.E. degree in information and computer sciences at Nagoya University. His current research interests include computer graphics and high performance computing.

Masao Okita is an Associate Professor in the Department of Computer Science, Graduate School of Information Science and Technology, Osaka University since 2009. He received the B.E. and M.E., and Ph.D. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 2001, 2003, and 2006, respectively. From 2006 to 2009, he worked for Acces Co., Ltd., as a section chief. His research interests include parallel processing, especially physiological simulation on supercomputers and GPU-accelerated machine learning.

Fumihiko Ino received the B.E., M.E., and Ph.D. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 1998, 2000, and 2004, respectively. He is currently a Professor in the Graduate School of Information Science and Technology at Osaka University. His research interests include parallel and distributed systems, software development tools, and performance evaluation.