An Algorithm for the All-pair Shortest Path Problem on Circular Trapezoid Graphs

Hirotoshi Honma[1*] Yoko Nakajima[2] Tsendsuren Urangoo[3] Yuto Tamori[4]

[1*,2] *Department of Creative Engineering, National Institute of Technology, Kushiro College, Japan*
[3,4] *Department of Information Engineering, National Institute of Technology, Kushiro College, Japan*

*Corresponding Author. E-mail address: honma@kushiro-ct.ac.jp

Received: 8 April 2020; Revised: 7 May 2020; Accepted: 22 May 2020
Published online: 25 June 2020

Abstract

The shortest path problem involves finding a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimized. This is a fundamental and important problem in graph theory, and it has been applied for solving numerous problems such as the minimum connected dominating set, minimum-weight circle-cover, and minimum cardinality Steiner set problems. The single-source shortest path problem involves finding the shortest paths between a given vertex and all other vertices. The all-pair shortest path problem involves the determination of the shortest graph distances between every pair of vertices in a given graph. In general, it is known that efficient sequential or parallel algorithms can be developed by restricting the classes of graphs. Numerous studies have been conducted on the shortest path problems on several intersection graphs. A circular trapezoid graph is a proper superclass of trapezoid graphs and circular-arc graphs. In this study, we present an O(n^2) time algorithm to solve the all-pair shortest path problem on circular trapezoid graphs.

Keywords: design and analysis of algorithms, shortest path, intersection graphs, circular trapezoid graphs

I. INTRODUCTION

The shortest path problem involves finding a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimized. This is a fundamental and important problem in graph theory, and it has been applied for solving numerous problems such as the minimum connected dominating set, minimum-weight circle-cover, and minimum cardinality Steiner set problems.

The single-source shortest path (SSSP) problem involves finding the shortest paths between a given vertex and all other vertices. A well-known solution to the SSSP problem was discovered by Dijkstra [1]. A sophisticated implementation can be found in [2]. The all-pair shortest path (APSP) problem involves the determination of the shortest graph distances between every pair of vertices in a given graph. A well-known algorithm for solving the APSP problem was developed by Floyd [3], who obtained the result based on a theorem proposed by Warshall [4]. The all-pair shortest path query problem does not find all pairs of shortest paths; instead, it is described as follows: First, a faster preprocessing algorithm is applied and a data structure is constructed. Then, a query on the length of the shortest path between any two vertices can be answered quite rapidly using the data structure. For other recent studies related to the shortest path problems, please refer to [5]-[8].

In general, it is known that efficient sequential or parallel algorithms can be developed by restricting the classes of graphs. Numerous studies have been conducted on the shortest path problems on several intersection graphs. For example, Hsu et al. developed an O(n) time algorithm for the SSSP problem on interval and circular-arc graphs [9]. Ibarra and Zheng demonstrated an optimal parallel algorithm for the SSSP problem for permutation graphs. The algorithm runs in O(log n) time using O(n/log n) processors [10]. Mondal et al. presented an O(n^2) time algorithm for the APSP problem on trapezoid graphs [11]. Barman et al. presented an efficient algorithm for the next-to-shortest path problem on trapezoid graphs; the algorithm runs in O(n^2) time [12].

Lin [13] introduced a circular trapezoid graph (CTG), which is a proper superclass of trapezoid graphs and circular-arc graphs. They showed that the maximum weighted independent set can be found in O(n^2 log log n) time on a CTG [13]. In this study, we present an O(n^2) time algorithm to solve the APSP problem on a CTG.

The rest of this paper is organized as follows: Section 2 defines CTGs and models and introduces the extended circular trapezoid model that is constructed from a circular trapezoid model. Moreover, we provide the definitions and notations used in this paper. Section 3 presents the properties of the shortest path on a CTG, which are useful for efficiently solving the APSP problem. Moreover, we describe our algorithm for solving the APSP problem and its complexity. Section 4 provides the benchmark experimental results. Finally, Section 5 concludes the paper.

II. DEFINITIONS

A. Circular Trapezoid Model and Graph

We describe a circular trapezoid model (CTM) before defining a circular trapezoid graph (CTG). A CTM consists of inner and outer circles, C_1 and C_2, with radii r_1 and r_2, respectively, where $r_1 < r_2$. Each circle is arranged counterclockwise with consecutive integer values 1, 2,..., 2n, where n is the number of trapezoids. Consider two arcs, A_1 and A_2, on C_1 and C_2, respectively.

Points *a* and *b* represent the first points encountered when traversing arc A_1 counterclockwise and clockwise, respectively. Similarly, points *c* and *d* represent the first points encountered when traversing arc A_2 counterclockwise and clockwise, respectively. A trapezoid is the region in circles C_1 and C_2 that lies between two non-crossing chords, *ac* and *bd*. Trapezoid CT_i is defined by four corner points, $[a_i, b_i, c_i, d_i]$. Without loss of generality, we assume that each trapezoid contains four corner points and all corner points are distinct. Each trapezoid, CT_i, is numbered in the ascending order of its corner points, a_i, i.e., $i < j$ if $a_i < a_j$. The geometric representation described above is referred to as a CTM. Figure 1 illustrates an example of a CTM *M*, with twelve trapezoids.

An example of the use of a CTM is for cities consisting of cityscapes that spread radially around facilities such as stations and rotaries. In this case, the CTM visually represents the relationships among communities (linkage of transportation networks, sharing of infrastructure facilities, etc.) and is applied to the optimization of city planning and facility arrangement. Table 1 shows the details of *M* as depicted in Figure 1.

Consider a fictitious line, *p*, that connects the points placed between 1 and 2*n* of C_1 and C_2. A trapezoid that intersects *p* is referred to as a feedback trapezoid. In Figure 1, CT_2 and CT_5 are feedback trapezoids. If a fictitious line, *p*', that connects a point on C_1 and a point on C_2 does not intersect any trapezoid in the CTM, a model obtained by opening the CTM along *p*' is equivalent to a normal trapezoid model. Then, the APSP problem can be solved by applying Mondal's algorithm [11] because it is the same as that of normal trapezoid graphs. We assume that any fictitious line, *p*', that connects a point on C_1 and a point on C_2 intersects at least one trapezoid.

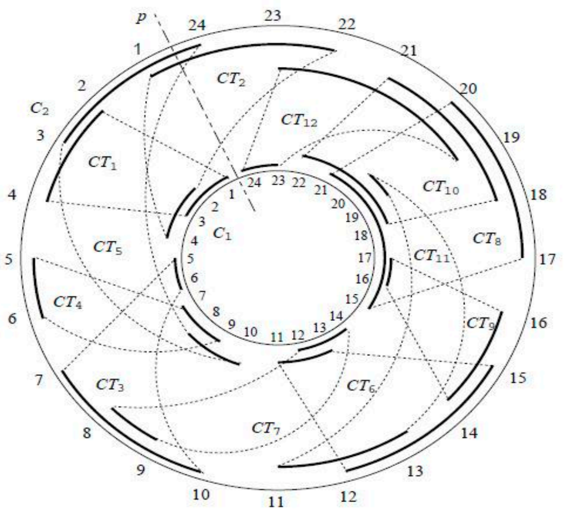

Figure 1 Circular trapezoid model *M*

A graph, *G*, is a CTG if it can be represented by the following CTM *M*: each vertex of the graph corresponds to a trapezoid, and two vertices are adjacent in *G* if and only if their corresponding trapezoids intersect. Figure 2 illustrates the CTG, *G*, corresponding to the CTM *M*, shown in Figure 1.

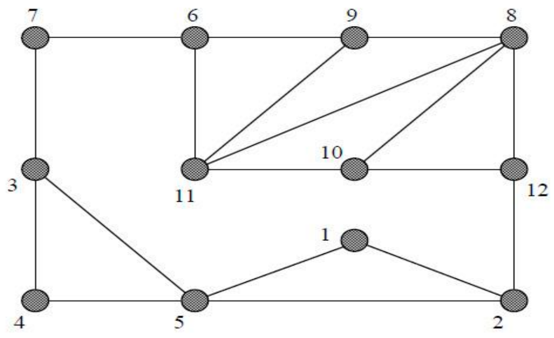

Figure 2 Circular trapezoid graph

Table 1 Detail of the CTM *M*

i	1	2	3	4	5	6	7	8	9	10	11	12
a_i	1	2	5	7	8	11	12	15	16	18	19	23
b_i	3	4	6	9	10	13	14	21	17	22	20	24
c_i	2	22	7	5	24	12	8	17	14	18	11	19
d_i	4	1	10	6	3	15	9	20	16	21	13	23

B. Extended Circular Trapezoid Model

Next, we introduce an extended circular trapezoid model (ECTM) constructed from the CTM. Let *n* be the number of trapezoids in CTM *M*. Consider a fictitious line, *p*, that connects the points placed between 1 and 2*n* of C_1 and C_2. First, the CTM is cut along fictitious line *p* and C_1 and C_2 are expanded into parallel horizontal lines (referred to as the top and bottom channels). Hereafter, to prevent confusion, we denote a trapezoid in the CTM and ECTM by CT_i and T_i, respectively. The above process is followed to construct an ECTM from the CTM. This process can be executed in O(*n*) time [14]. Figure 3 illustrates the ECTM *EM*, constructed from the CTM *M*, shown in Figure 1. Table 2 shows the details of *EM*.

Honma et al. presented a procedure that constructs *EM* from *M* [14].

Lemma 1. [14] *An ECTM EM, corresponding to a given CTM M, is constructed in O(n) time.* □

Efficient algorithms have been developed to address various problems concerning noncircular intersection graphs (interval, permutation, trapezoid, etc). However, in general, the problems for circular intersection graphs tend to be more difficult than those for noncircular intersection graphs. One of the reasons for this difficulty is that we cannot uniquely determine the starting position of an algorithm for circular intersection graphs owing to the existence of feedback elements. However, this position can be determined for noncircular intersection graphs. For several problems, we can develop the circular versions of existing algorithms by constructing the extended intersection models (such as the ECTM) of the problems. We can uniquely determine the starting position of an algorithm and partially apply the algorithms of noncircular versions using these models. For instance, this method has been utilized to develop efficient algorithms for the shortest path query problem [9], [15] and articulation vertex problem [16] in circular-arc graphs, the maximum clique and chromatic number problems [17], spanning forest problem [18], and articulation problem [19] in circular permutation graphs, and the spanning tree problem [14] and hinge vertex problem [20] in CTGs.

C. Definitions

We introduce the definitions and notations that are used in our algorithm. Let *G* be a CTG corresponding to CTM *M* and *EM* be an ECTM constructed from *M*.

We define *rt*(*i*), *rb*(*i*), *lt*(*i*), and *lb*(*i*) as follows: Here, the set (including *i*) of all trapezoids that intersect T_i in *EM* is denoted by *N*[*i*].

- $rt(i) = k$ such that $b_k = \max\{ b_j \mid j \in N[i] \}$,
- $rb(i) = k$ such that $d_k = \max\{ d_j \mid j \in N[i] \}$,
- $lt(i) = k$ such that $a_k = \min\{ a_j \mid j \in N[i] \}$,
- $rb(i) = k$ such that $c_k = \min\{ c_j \mid j \in N[i] \}$.

TABLE 2 shows the details of *rt*(*i*), *rb*(*i*), *lt*(*i*), and *rb*(*i*), and the ECTM is illustrated in Figure 3.

Let $tp_1 < tp_2$ be the coordinates on the top channel and $bp_1 < bp_2$ be the coordinates on the bottom channel in *EM*. We define sets $A[tp_1, tp_2]$, $C[bp_1, bp_2]$, $B[tp_1, tp_2]$, and $D[bp_1, bp_2]$ as follows:

- $A[tp_1, tp_2] = \{ k \mid tp_1 < a_k < tp_2 \}$,
- $C[bp_1, bp_2] = \{ k \mid bp_1 < c_k < bp_2 \}$,
- $B[tp_1, tp_2] = \{ k \mid tp_1 < b_k < tp_2 \}$,
- $D[bp_1, bp_2] = \{ k \mid bp_1 < d_k < bp_2 \}$.

r-tree(*i*) is a tree with root *i*, and it represents the shortest path when visiting all vertices from *i* to the right on *EM*. Similarly, l-tree(*i*) is a tree with root *i*, and it represents the shortest path when visiting all vertices from *i* to the left on *EM*. $dis_r(i, j)$ and $dis_l(i, j)$ are the distances from root *i* to vertex *j* on r-tree(*i*) and l-tree(*i*), respectively.Figures 4 and 5 show the examples of r-tree(6) and l-tree(6), respectively. In this example, we have $dis_r(6, 5) = 5$ and $dis_l(6, 5) = 3$.

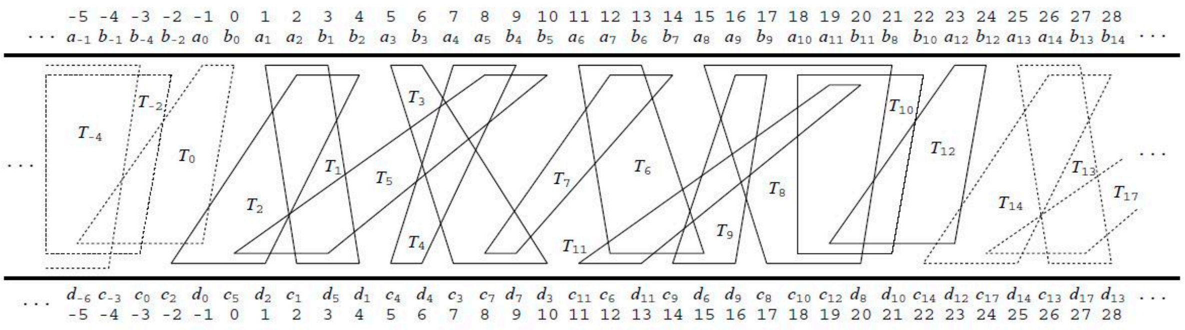

Figure 3 Extended circular trapezoid model *EM*

Table 2 Detail of the ECTM *EM*

i	-11	-10	...	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...	23	24
a_i	-23	-22	...	-8	-6	-5	-1	1	2	5	7	8	11	12	15	16	18	19	23	25	26	29	31	...	43	47
b_i	-21	-20	...	-7	-2	-4	0	3	4	6	9	10	13	14	21	17	22	20	24	27	28	30	33	...	44	48
c_i	-22	-26	...	-10	-6	-13	-5	2	-2	7	5	0	12	8	17	14	18	11	19	26	22	31	29	...	35	43
d_i	-20	-23	...	-8	-3	-11	-1	4	1	10	6	3	15	9	20	16	21	13	23	28	25	34	30	...	37	47
$rt(i)$	-7	-7	...	-4	0	-2	2	5	5	5	5	5	11	7	12	8	12	10	14	17	17	17	17	...	22	24
$rb(i)$	-11	-11	...	-4	0	-2	2	1	1	3	3	3	9	6	12	8	12	10	14	13	13	15	15	...	22	24
$lt(i)$	-11	-11	...	-6	-4	-6	-4	1	0	3	3	3	6	3	8	6	8	6	8	13	12	15	15	...	18	20
$lb(i)$	-10	-10	...	-1	-1	-1	-4	2	0	5	5	2	7	3	11	11	11	11	8	14	12	17	17	...	23	20

III. RESEARCH METHODOLOGY

A. Useful Properties

In this section, we present some useful properties for constructing an algorithm for computing the shortest path lengths between the all-pair vertices of CTG *G*. We describe a few lemmas that are useful for constructing the algorithm for the APSP problem in CTGs.

Lemma 2. *r-tree(i) and l-tree(i), 1 ≤ i ≤ n, can be constructed in O(n^2) time using Mondal's algorithm [11] when the corresponding ECTM is given as an input.* □

Mondal's algorithm [11] can solve the APSP problem of a normal trapezoid graph in O(n^2) time by constructing shortest path trees. The ECTM is constructed from the CTM, and it is a geometric figure model similar to the trapezoid model. We can obtain r-tree(*i*) and l-tree(*i*), 1 ≤ *i* ≤ *n*, in O(n^2) time by applying Mondal's algorithm to the ECTM.

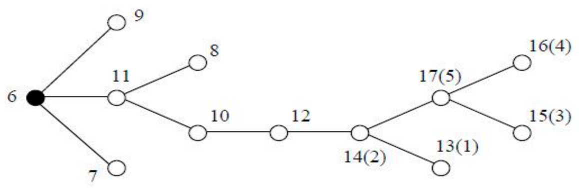

Figure 4 Example of r-tree(6)

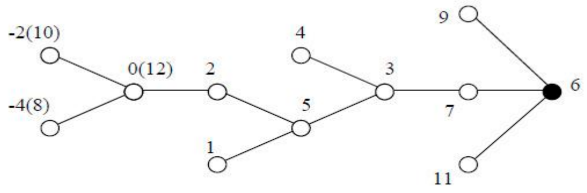

Figure 5 Example of l-tree(6)

Lemma 3. *For any pair of vertices, i and j, on CTG G, distance dis(i, j) is obtained from min{ dis$_r$(i, j), dis$_l$(i, j) }.*

We assume two nonintersecting trapezoids, CT_i and CT_j, on the CTM. There are two paths from CT_i to CT_j on the CTM. One follows a trapezoid counterclockwise from CT_i to CT_j, and the other follows a trapezoid clockwise. r-tree(i) and l-tree(i) are the shortest path trees that can be obtained when moving from CT_i to another trapezoid in the counterclockwise and clockwise directions, respectively.

Therefore, the shortest path length between vertices i and j is min{ $dis_r(i, j)$, $dis_l(i, j)$ }.

Figure 6 shows an example of the shortest path from vertex 6 to 5 on the CTG. In this example, < 6, 11, 10, 12, 2, 5 > is the shortest counterclockwise path from vertex 6 on the CTM, and we obtain the length, $dis_r(6, 5) = 5$, from r-tree(6) shown in Figure 4. Similarly, < 6, 7, 3, 5 > is the shortest clockwise path from vertex 6 on the CTM, and we obtain the length, $dis_l(6, 5) = 3$, from l-tree(6) shown in Figure 5. Therefore, the shortest path length from 6 to 5 on this CTG is 3.

B. Algorithm APSP-CTG and Its Complexity

In this section, we present the APSP-CTG algorithm for computing the shortest path lengths between the all-pair vertices of a CTG, G. In this algorithm, we define a function for normalization, nor(i), which is expressed as

$$\mathrm{nor}(i) = \begin{cases} i - n & \text{if } i > n, \\ i + n & \text{if } i < 1, \\ i & \text{if } 1 \leqslant i \leqslant n, \end{cases}$$

for a trapezoid, T_i, in the ECTM. The value of nor(i) is the vertex number in the CTG corresponding to the copy of trapezoid T_i in the ECTM. For the example shown in Figure 3, we have nor(-4)=10 and nor(17)=5.

Now, we concisely describe the outline of our algorithm (Algorithm APSP-CTG) and analyze its complexity.

Algorithm~APSP-CTG considers a CTM M, as an input. In Step 1, we construct an ECTM EM, from M using Honma's algorithm [14] (Figure 2). This step can be performed in O(n) time. In Step 2, we compute $rt(i)$, $rb(i)$, $lt(i)$, and $lb(i)$, for $1\text{-}n \leq i \leq 2n$, using prefix computation [21]. This process can execute in O(n) time. Steps 3 and 4 construct r-tree(i) and l-tree(i), for $1 \leq i \leq n$, respectively. These steps can be executed in O(n^2) time using Mondal's algorithm [11]. Figures 4 and 5 show the examples of r-tree(6) and l-tree(6), respectively. Step 5

Algorithm 1: APSP-CTG

```
Input: Corner points $a_i, b_i, c_i, d_i$ for each trapezoid in a CTM $M$.
Output: All-pair shortest path matrix $D$ for the CTG $G$.
(Step 1) /* Construction of $EM$ from $M$ */
Construct an ECTM $EM$ from $M$ using Honma's algorithm [10];
(Step 2) /* Preprocessing */
for $1 \leqslant i \leqslant n$ do
    Compute $rt(i)$, $rb(i)$, $lt(i)$, and $lb(i)$ on $EM$;
(Step 3) /* Construction of r-tree(i) */
Note that $N[i]$ is a set $\{k \mid T_k$ intersects with $T_i$ on $EM\}$ ;
for $1 \leqslant i \leqslant n$ do
    r-tree($i$) := $\emptyset$ ;
    for $1 \leqslant j \leqslant n$ do $v[j]$ := '0' ;
    $v[i]$ := '1' ;
    for $k \in N[i]$ do
        r-tree($i$) := r-tree($i$) $\cup$ $(i, \mathrm{nor}(k))$ ; /* Add an edge */
        $v[\mathrm{nor}(k)]$ := '1' ;
    $tp_1 := b_i$, $tp_2 := b_{rt(i)}$ ;
    $bp_1 := d_i$, $bp_2 := b_{rb(i)}$ ;
    $i_t := i_b := i$ ;
    while there exists $j$ such as $v[j]$ = '0' do
        for $k \in A[tp_1, tp_2]$, $v[\mathrm{nor}(k)]$ = '0' do
            r-tree($i$) := r-tree($i$) $\cup$ $(i_t, \mathrm{nor}(k))$ ; /* Add an edge */
            $v[\mathrm{nor}(k)]$ := '1' ;
        for $k \in C[bp_1, bp_2]$, $v[\mathrm{nor}(k)]$ = '0' do
            r-tree($i$) := r-tree($i$) $\cup$ $(i_b, \mathrm{nor}(k))$ ; /* Add an edge */
            $v[\mathrm{nor}(k)]$ := '1' ;
        $tp'_2 := \max\{\ b_j \mid j \in A[tp_1, tp_2]\}$ ;
        $tp''_2 := \max\{\ b_j \mid j \in C[bp_1, bp_2]\}$ ;
        $bp'_2 := \max\{\ d_j \mid j \in A[tp_1, tp_2]\}$ ;
        $bp''_2 := \max\{\ d_j \mid j \in C[bp_1, bp_2]\}$ ;
        $tp_1 := tp_2$, $tp_2 := \max\{tp'_2, tp''_2\}$ ;
        $bp_1 := bp_2$, $bp_2 := \max\{bp'_2, bp''_2\}$ ;
        $i_t := k$ such that $b_k = tp_2$ ;
        $i_b := k$ such that $b_k = bp_2$ ;
(Step 4) /* Construction of l-tree(i) */
for $1 \leqslant i \leqslant n$ do
    l-tree($i$) := $\emptyset$ ;
    for $1 \leqslant j \leqslant n$ do $v[j]$ := '0' ;
    $v[i]$ := '1' ;
    for $k \in N[i]$ do
        l-tree($i$) := l-tree($i$) $\cup$ $(i, \mathrm{nor}(k))$ ; /* Add an edge */
        $v[\mathrm{nor}(k)]$ := '1' ;
    $tp_1 := a_{lt(i)}$, $tp_2 := a_i$ ;
    $bp_1 := c_{lb(i)}$, $bp_2 := c_i$ ;
    $i_t := i_b := i$ ;
    while there exists $j$ such as $v[j]$ = '0' do
        for $k \in B[tp_1, tp_2]$, $v[\mathrm{nor}(k)]$ = '0' do
            l-tree($i$) := l-tree($i$) $\cup$ $(i_t, \mathrm{nor}(k))$ ; /* Add an edge */
            $v[\mathrm{nor}(k)]$ := '1' ;
        for $k \in D[bp_1, bp_2]$, $v[\mathrm{nor}(k)]$ = '0' do
            l-tree($i$) := l-tree($i$) $\cup$ $(i_b, \mathrm{nor}(k))$ ; /* Add an edge */
            $v[\mathrm{nor}(k)]$ := '1' ;
        $tp'_1 := \min\{\ a_j \mid j \in B[tp_1, tp_2]\}$ ;
        $tp''_1 := \min\{\ a_j \mid j \in D[bp_1, bp_2]\}$ ;
        $bp'_1 := \min\{\ c_j \mid j \in B[tp_1, tp_2]\}$ ;
        $bp''_1 := \min\{\ c_j \mid j \in D[bp_1, bp_2]\}$ ;
        $tp_2 := tp_1$, $tp_1 := \min\{tp'_1, tp''_1\}$ ;
        $bp_2 := bp_1$, $bp_1 := \min\{bp'_1, bp''_1\}$ ;
        $i_t := k$ such that $b_k = tp_1$ ;
        $i_b := k$ such that $b_k = bp_1$ ;
(Step 5) /* Construction of APSP matrix D */
Note that $\mathrm{dis}_r(i, j)$ and $\mathrm{dis}_l(i, j)$ are the distances from root $i$ to $j$ on
r-tree($i$) and l-tree($i$), respectively.
for $1 \leqslant i \leqslant n$ do
    for $1 \leqslant j \leqslant n$ do
        $D[i, j] := \min\{\mathrm{dis}_r(i, j), \mathrm{dis}_l(i, j)\}$ ;
```

constructs APSP matrix D. From Lemma 3, the distance, $dis(i, j)$, is computed using min{ $dis_r(i, j)$, $dis_l(i, j)$ }. This step can be executed in $O(n^2)$ time.

Thus, we obtain the following theorem:

Theorem 1. *Algorithm APSP-CTG computes the all-pair shortest path for a CTG in $O(n^2)$ time by considering its CTM M, as an input.* □

IV. RESULTS AND DISCUSSION

We actually implemented the program and performed a benchmark test of the execution speed. In the APSP problem, we compared our well-known Floyd-Warshall method with our method. Ten graphs each with 5,000 and 10,000 nodes were created and averaged for each processing time. In processing time benchmarks, graph samples with node sizes of 5,000 and 10,000 are considered adequate. Moreover, in order to absorb the deviation of the shape of graph and the number of edges, the average of 10 samples was calculated.

Table 3 shows the benchmark experimental results. The time complexty of Floyd-Warshall is $O(n^3)$, while the complexity of our algorithm is $O(n^2)$. The benchmark values show that our algorithm achieves significant speedup.

Floyd-Warshall Algorithm solves the APSP problem using dynamic programming. As often practiced in dynamic programming, the problem is divided into smaller subproblems which are then solved to obtain intermediate results to be used in the overall solution. In the Floyd-Warshall algorithm, it uses the adjacent matrix of the graph given as input and requires $O(n^3)$ computation time. On the other hand, our algorithm uses the intersection model as an input and holds the shortest distance between verticess in a tree structure, so that the computational complexity can be suppressed to $O(n^2)$.

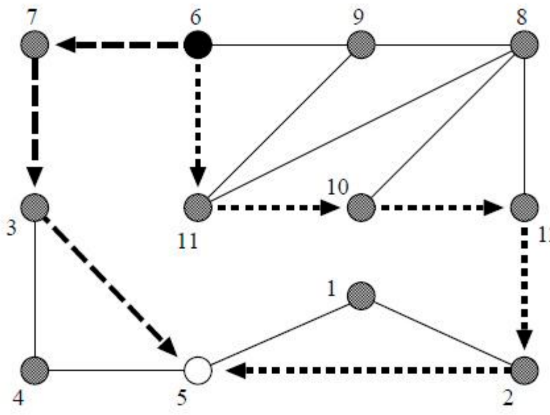

Figure 6 Shortest path from vertex 6 to 5

$$D = \begin{pmatrix}
0 & 1 & 2 & 2 & 1 & 4 & 3 & 3 & 4 & 3 & 4 & 2 \\
1 & 0 & 2 & 2 & 1 & 4 & 3 & 2 & 3 & 2 & 3 & 1 \\
2 & 2 & 0 & 1 & 1 & 2 & 1 & 4 & 3 & 4 & 3 & 3 \\
2 & 2 & 1 & 0 & 1 & 3 & 2 & 4 & 4 & 4 & 4 & 3 \\
1 & 1 & 1 & 1 & 0 & 3 & 2 & 3 & 4 & 3 & 4 & 2 \\
4 & 4 & 2 & 3 & 3 & 0 & 1 & 2 & 1 & 2 & 1 & 3 \\
3 & 3 & 1 & 2 & 2 & 1 & 0 & 3 & 2 & 3 & 2 & 4 \\
3 & 2 & 4 & 4 & 3 & 2 & 3 & 0 & 1 & 1 & 1 & 1 \\
4 & 3 & 3 & 4 & 4 & 1 & 2 & 1 & 0 & 2 & 1 & 2 \\
3 & 2 & 4 & 4 & 3 & 2 & 3 & 1 & 2 & 0 & 1 & 1 \\
4 & 3 & 3 & 4 & 4 & 1 & 2 & 1 & 1 & 1 & 0 & 2 \\
2 & 1 & 3 & 3 & 2 & 3 & 4 & 1 & 2 & 1 & 2 & 0
\end{pmatrix}$$

Table 3 Benchmark Results

Size	Genaral Algorithm	Our Algorithm
5,000	58.91 [sec]	1.44 [sec]
10,000	502.46 [sec]	6.02 [sec]

Processor: Intel Core i7-8700 3.2GHz
Memory: 32GB, Compiler: Borland C++ 5.5.1

V. CONCLUSION

In this study, we proposed Algorithm APSP-CTG, which operates in $O(n^2)$ time to compute the APSP problem in a CTG. Our algorithm partially uses Mondal's [11] and Honma's algorithms [14]. The APSP problem is a fundamental problem that is used to solve numerous problems in graph theory. For this reason, we believe that this paper has merits from a theoretical and algorithmic point of view. Reducing the complexity of the algorithm and extending the results to other graphs will be addressed in future work.

ACKNOWLEDGEMENT

We express many thanks to anonymous referees for their valuable advices on the theory of our attacks and their helpful editorial comments. This work was partially supported by JSPS KAKENHI Grant Number 19K11834 and 17K00324, and Cooperative Education/Research Project between Toyohashi University of Technology and National Institute of Technology.

REFERENCES

[1] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematlk*, vol. 1, pp. 269–271, 1959.

[2] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, no. 3, pp. 596–615, Jul. 1987.

[3] R. W. Floyd, "Algorithm 97: Shortest path," *Communications of the ACM*, vol. 5, no. 6, pp. 344-348, Jun. 1962.

[4] S. Warshall, "A theorem on boolean matrices," *J. ACM*, vol. 9, no.1, pp. 11–12, Jan. 1962.

[5] Y. Kobayashi and R. Sako, "Two disjoint shortest paths problem with non-negative edge length," *Operations Research Letters*, vol. 47, no. 1, pp. 66–69, Jan. 2019.

[6] C. Glacet, N. Hanusse, D. Ilcinkas, and C. Johnen, "Disconnected components detection and rooted shortest-path tree maintenance in networks," *J. Parallel and Distributed Computing*, vol. 132, pp. 299–309, Oct. 2019.

[7] M. Debski, K. J. Szaniawski and Z. Lonc, "Bundling all shortest paths," *Discrete Applied Mathematics*, vol. 277, pp. 82–91, Apr. 2020.

[8] L. Di Puglia Pugliese, D. Ferone, P. Festa, and F. Guerriero, "Shortest path tour problem with time windows," *European Journal of Operational Research*, vol. 282, no. 1, pp. 334–344, Apr. 2020.

[9] F. R. Hsu, K. Shan, H. S. Chao, and R. C. Lee, "Some optimal parallel algorithms on interval and circular-arc graphs," *J. Inf. Sci. Eng.*, vol. 21, no. 3, pp. 627–642, May 2005.

[10] O. H. Ibarra and Q. Zheng, "An optimal shortest path parallel algorithm for permutation graphs," *J. Parallel and Distributed Computing*, vol. 24, no. 1, pp. 94–99, Jan. 1995.

[11] S. Mondal, M. Pal, and T. K. Pal, "An optimal algorithm for solving all-pairs shortest paths on trapezoid graphs," *Int. J. Computational Engineering Science*, vol. 3, no. 2, pp. 103–116, 2002.

[12] S. C. Barman, S. Mondal, and M. Pal, "An efficient algorithm to find next-to-shortest path on trapezoid graphs," *Advances in Applied Mathematical Analysis*, vol. 2, no. 2, pp. 97–107, 2007.

[13] W. L. Lin, "Circular and circle trapezoid graphs," *J. Sci. Eng. Tech.*, vol. 2, no. 2, pp. 11–17, 2006.

[14] H. Honma, Y. Nakajima, Y. Aoshima, and S. Masuyama, "A lineartime algorithm for constructing a spanning tree on circular trapezoid graphs," *IEICE Trans. Fundamentals*, vol. E96-A, no. 6, pp. 1051–1058, Jun. 2013.

[15] D. Chen, D. T. Lee, R. Sridhar, and C. Sekharam, "Solving the all-pair shortest path query on interval and circular-arc graphs," *Networks*, vol. 31, no. 4, pp. 249–258, 1998.

[16] T. W. Kao and S. J. Horng, "Optimal algorithms for computing articulation points and some related problems on a circular-arc graph," *Parallel Computing*, vol. 21, no. 6, pp. 953–969, Jun. 1995.

[17] R. D. Lou and M. Sarrafzadeh, "Circular permutation graph family with applications," *Discrete Applied Mathematics*, vol. 40, no. 3, pp. 433–457, Dec. 1992.

[18] H. Honma, S. Honma, and S. Masuyama, "An optimal parallel algorithm for constructing a spanning tree on circular permutation graphs," *IEICE Transactions on Information and Systems*, vol. E92-D, no. 2, pp. 141–148, Feb. 2009.

[19] H. Honma, K. Abe, Y. Nakajima, and S. Masuyama, "Linear time algorithms for finding articulation and hinge vertices of circular permutation graphs," *IEICE Trans. Inf. & Syst.*, vol. E96-D, no. 3, pp. 419–425, Mar. 2013.

[20] H. Honma, Y. Nakajima, and S. Masuyama, "An algorithm for hinge vertex problem on circular trapezoid graphs," *Journal of Information Processing*, vol. 25, pp. 945–948, Dec. 2017.

[21] D. Bera, M. Pal, and T. K. Pal, "An efficient algorithm for finding all hinge vertices on trapezoid graphs," *Theory of Computing Systems*, vol. 36, no. 1, pp. 17–27, Feb. 2003.