A Linear-Time Algorithm for Finding a Paired 2-Disjoint Path Cover in the Cube of a Connected Graph

Insung Ihm[a], Jung-Heum Park[b,*]

[a]*Department of Computer Science and Engineering*
Sogang University, Seoul, Korea
[b]*School of Computer Science and Information Engineering*
The Catholic University of Korea, Bucheon, Korea

Abstract

For a connected graph $G = (V(G), E(G))$ and two disjoint subsets of $V(G)$ $A = \{\alpha_1, \ldots, \alpha_k\}$ and $B = \{\beta_1, \ldots, \beta_k\}$, a paired (many-to-many) k-disjoint path cover of G joining A and B is a vertex-disjoint path cover $\{P_1, \ldots, P_k\}$ such that P_i is a path from α_i to β_i for $1 \leq i \leq k$. In the recent paper [Disjoint Path Covers in Cubes of Connected Graphs, Discrete Mathematics 325 (2014) 65–73], Park and Ihm presented a necessary and sufficient condition for a paired 2-disjoint path cover joining two vertex sets to exist in the cube of a connected graph. In this paper, we propose an $O(|V(G)| + |E(G)|)$-time algorithm that actually finds such a paired 2-disjoint path cover. In particular, we show that, in order to build a desired disjoint path cover, it is sufficient to consider only the edges of a carefully selected spanning tree of the graph and at most one additional edge not in the tree, which allows an efficient linear-time algorithm.

Keywords: Disjoint path cover, cube of graph, Hamiltonian path, spanning tree, unicyclic graph, linear-time algorithm.

1. Introduction

Given two vertices v and w of a finite simple undirected graph $G = (V(G), E(G))$, a *path* P in G from v to w is a sequence $\langle u_0, u_1, \ldots, u_n \rangle$ of

*Corresponding author
Email addresses: `ihm@sogang.ac.kr` (Insung Ihm), `j.h.park@catholic.ac.kr` (Jung-Heum Park)

distinct vertices of G such that $u_0 = v$, $u_n = w$, and $(u_i, u_{i+1}) \in E(G)$ for $i = 0, \ldots, n-1$. If $n \geq 2$ and $(u_0, u_n) \in E(G)$, the sequence $\langle u_0, u_1, \ldots, u_n, u_0 \rangle$ is called a *cycle*. A *path cover* of G is a set of paths in G such that every vertex of G is contained in at least one path. A *vertex-disjoint path cover*, or simply a *disjoint path cover*, of G is a special kind of path cover in which every vertex of G is covered by exactly one path. For a given positive integer k, let $A = \{\alpha_1, \ldots, \alpha_k\}$ and $B = \{\beta_1, \ldots, \beta_k\}$ be two disjoint subsets of $V(G)$. Then, a disjoint path cover $\{P_1, \ldots, P_k\}$ of G is said to be a *paired (many-to-many) k-disjoint path cover* (a *paired k-DPC* in short) joining A and B if P_i is a path from α_i to β_i for $i = 1, \ldots, k$ [34]. In addition, the k-disjoint path cover is regarded as *unpaired* if P_i is allowed to be a path from α_i to $\beta_{\sigma(i)}$ for some permutation σ on $\{1, \ldots, k\}$. Here, the vertices in A and B are often called *sources* and *sinks* respectively, and *terminal vertices* (or simply *terminals*) collectively. Simpler variants of these disjoint path covers have also been investigated in the graph theory community, and readers are recommended to refer to related literature such as [18, 31, 34] for more details.

Clearly, the paired k-DPC is more restrictive than the unpaired k-DPC in the sense that the paired one is just one of up to exponentially many unpaired one joining A and B. Therefore, a solution of a problem for the paired k-DPC often easily leads to an unpaired k-DPC solution for the same problem. An interesting question regarding the paired k-DPC is how to determine whether a desired disjoint path cover exists for a given connected graph G and two terminal sets A and B. In the recent paper [32], Park and Ihm answered this question partially for a specific class of graphs where the exact condition on the terminal sets A and B that allow a paired 2-DPC in the cube of G was presented where the *cube* of G, denoted by G^3, is the graph with the same vertex set $V(G)$, having an edge between two vertices if and only if there exists a path of length at most 3 in G joining them.

Naturally, the next question is how efficiently such a disjoint path cover can be found when it exists. In this paper, we propose a linear-time algorithm that finds a paired 2-DPC in the cube of a connected graph joining two terminal sets. In particular, we prove an interesting fact that it suffices to examine only the edges of a carefully selected spanning tree of the graph and possibly one more edge not in the tree, to find a desired paired 2-DPC efficiently. Trivially, this linear-time algorithm is also effective to building an *unpaired* 2-DPC if exists because a solution can be found by applying the algorithm at most twice over the two permutations on the two-element terminal sets.

This paper is organized as follows. In the rest of this section, we briefly discuss related concepts and review relevant previous work. Section 2 presents preliminaries that will be referred to frequently throughout the paper. Then, Section 3 describes a linear-time algorithm that finds a paired 2-DPC in the cube of a tree joining two terminal sets, which will then be used effectively in Section 4 to build our main algorithm that finds the disjoint path cover in the cube of an arbitrary connected graph. Finally, the paper is concluded in Section 5.

1.1. Vertex connectivity

The existence of a disjoint path cover in a graph is closely related to the concept of vertex connectivity: Menger's theorem states the connectivity of a graph in terms of the number of disjoint paths joining two distinct vertices, whereas the Fan Lemma states the connectivity of a graph in terms of the number of disjoint paths joining a vertex to a set of vertices [1]. Moreover, it can be shown that a graph is k-connected if and only if it has k disjoint paths joining two arbitrary vertex sets of size k each, in which a vertex that belongs to both sets is counted as a valid path.

When a graph does not have a disjoint path cover of desired kind, it is natural to consider an augmented graph with higher connectivity. A simple way of increasing the connectivity is to raise a graph to a power: Given a positive integer d, the d-th power G^d of G is defined as a graph with the same vertex set $V(G)$ and the edge set that is augmented in such a way that two vertices of G^d are adjacent if and only if there exists a path of length at most d in G joining them. In particular, the graph G^2 is called the *square* of G, while G^3 is said to be the *cube* of G.

1.2. Disjoint path covers

The paired/unpaired (many-to-many) k-disjoint path cover has two simpler variants: The *one-to-many k-DPC* for $A = \{\alpha\}$ and $B = \{\beta_1, \ldots, \beta_k\}$ is a disjoint path cover made of k paths, each joining a pair of source α and sink β_i for $i \in \{1, \ldots, k\}$; The *one-to-one k-DPC* for $A = \{\alpha\}$ and $B = \{\beta\}$ is a disjoint path cover each of whose paths joins an identical pair of source α and sink β. The paths in the one-to-many k-DPC or in the one-to-one k-DPC may share a source and/or a sink and thus are pairwise internally disjoint.

The disjoint path cover problem has been studied for several classes of graphs: hypercubes [6, 9, 13, 15], recursive circulants [18, 19, 34, 35], hypercube-like graphs [34, 35], and grid graphs [33]. The structure of the cubes of connected graphs was investigated with respect to single-source 3-disjoint path covers [31]. The problem was also investigated in view of a full utilization of nodes in interconnection networks [34]. It was shown that deciding the existence of a one-to-one, one-to-many, or many-to-many k-DPC in a general graph, joining given sets of sources and sinks, is NP-complete for any fixed $k \geq 1$ [34, 35].

1.3. Strong Hamiltonian properties

The method for finding a disjoint path cover can easily be used for finding a Hamiltonian path (or cycle) due to its natural relation to the Hamiltonicity of graph. For instance, a Hamiltonian path between two distinct vertices in a graph G is in fact a 1-DPC of G joining the vertices. An Hamiltonian s–t path that passes a prescribed edge (x, y) with $\{s, t\} \cap \{x, y\} = \emptyset$ can also be found by solving the corresponding unpaired or paired 2-DPC problem [35]. While the unpaired version would be easier to tackle than the paired one, the difference is that the direction between x and y in the path may not be enforced through the unpaired 2-DPC. For more discussion on the Hamiltonian paths (or cycles) passing through prescribed edges, refer to, for example, [2, 8].

The cube of a connected graph with at least four vertices is 1-Hamiltonian, i.e., it is Hamiltonian and remains so after the removal of any one vertex, as Chartrand and Kapoor showed [5]. Sekanina [37] and Karaganis [16] independently proved that the cube of a connected graph is Hamiltonian-connected. Whether the cube is 1-Hamiltonian-connected, i.e. it still remains Hamiltonian-connected after the removal of any one vertex, was characterized for trees by Lesniak [23] and for connected graphs by Schaar [36]. Characterizations of connected graphs whose cubes are p-Hamiltonian for $p \leq 3$ were also made in [20, 36], and strong Hamiltonian properties of the cube of a 2-edge connected graph were studied in [30]. For discussion on the Hamiltonicity of the square of a graph, refer to, for example, [4, 7, 11, 12, 29].

1.4. Paired 2*-disjoint path covers of a connected graph*

While this paper is about a paired 2-DPC in the cube of a connected graph, it deserves to mention a relation between the presented algorithm and the more challenging problem of finding a paired 2-DPC of the graph itself. If a graph G is the d-th power of another graph H, i.e. $G = H^d$, then H is said to be a d-th *root* of G. In particular, we call such a graph H a *square root* of G if $d = 2$, and a *cube root* of G if $d = 3$. Mukhopadhyay characterized graphs that have a square root [28], and Escalante *et al.* then characterized graphs with a d-th root [10]. In general, deciding if a given graph has a d-th root is difficult, and it has been proven that recognizing squares of general graphs [27]/chordal graphs [22] and the d-th powers of bipartite graphs for $d \geq 3$ [21] are all NP-complete.

On the other hand, some classes of graphs for which the root finding problem is polynomially solvable have been investigated in the literature. Lin and Skiena presented a linear-time algorithm for recognizing squares of trees [24]. The general root finding problem for an arbitrary degree ($d \geq 3$) was solved over the trees in polynomial time [17] and then in linear time [3]. Lau and Corneil [22] gave a polynomial-time algorithm to recognize the d-th powers of proper interval graphs for $d \geq 2$. (For more information on the root finding problem, refer to the related work including [21, 25, 26].)

Note that if a graph G has a cube root H and, moreover, H could be found in time polynomial to the size of G, then the paired 2-DPC problem on $G\,(= H^3)$ would be polynomially solvable through our algorithm developed in this paper.

2. Preliminaries

2.1. The condition for a paired 2-DPC to exist in the cube of a graph

A *cut vertex* and *bridge* of a graph are respectively a vertex and an edge whose removal increases the number of connected components. A bridge, also known as a *cut-edge*, is said to be *nontrivial* if none of its two end-vertices is of degree one. A vertex of the graph is then called a *pure bridge vertex* if each of its incident edges is a nontrivial bridge. In addition, a pair of two adjacent vertices is called a *pure bridge pair* if both vertices are pure bridge vertices (refer to Figure 1 for a pictorial description of the pure bride vertex and pure bridge pair).

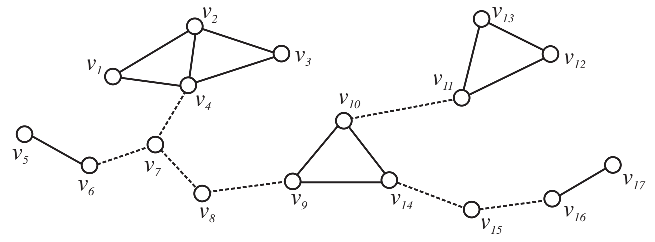


Figure 1: In this connected graph, the seven nontrivial bridges are marked in dotted lines. There are three pure bridge vertices v_7, v_8, and v_{15}, and one pure bridge pair $\{v_7, v_8\}$ [32].

Based on this notion, Park and Ihm presented a necessary and sufficient condition for the cube of a connected graph to have a paired 2-DPC joining two arbitrary terminal sets [32]. Interestingly, it was shown that two specific kinds of configurations between the four terminal vertices must be avoided to have a desired paired 2-DPC, as stated in the following theorem:

Theorem 1 (Park and Ihm [32]). *Let $A = \{\alpha_1, \alpha_2\}$ and $B = \{\beta_1, \beta_2\}$ be terminal sets of a connected graph G with at least four vertices. The cube G^3 has a paired 2-DPC joining A and B if and only if*

- ***C1:*** *$\{\alpha_1, \alpha_2, \beta_1, \beta_2\} \not\subseteq N_G[v]$ for any pure bridge vertex v of G, and*
- ***C2:*** *$\{\alpha_i, \beta_i\}$ is not a pure bridge pair of G such that $\{\alpha_{3-i}, \beta_{3-i}\} \subseteq N_G(\{\alpha_i, \beta_i\})$ for each $i = 1, 2$.*

Here, $N_G(v)$ and $N_G[v]$ for a vertex v of a graph G respectively represents the open and closed neighborhoods of v, i.e. $N_G(v) = \{u \in V(G) : (u, v) \in E(G)\}$ and $N_G[v] = N_G(v) \cup \{v\}$. These definitions are also naturally extended to a vertex set $X \subseteq V(G)$ in which $N_G(X) = \bigcup_{v \in X} N_G(v) \setminus X$ and $N_G[X] = N_G(X) \cup X$. Note that the validity of the two conditions, *C1* and *C2*, can be checked basically by identifying (nontrivial) bridges of the graph, which can easily be done in $O(|V(G)| + |E(G)|)$ time based on the well-known biconnected component algorithm [14].

2.2. Finding a Hamiltonian path in the cube of a tree

The fact that a Hamiltonian cycle can always be found in the cube of a connected graph in linear time is due to the theorem by Lin and Skiena [24].

Theorem 2 (Lin and Skiena [24]). *Given a connected graph G with $n\,(\geq 3)$ vertices and m edges and an integer $k \geq 3$, we can find a Hamiltonian cycle in G^k in $O(n + m)$ time.*

In the proof of the theorem, a Hamiltonian cycle in G^k is in fact built by first finding a spanning tree T of G and then finding a Hamiltonian path in T^3 between the two end-vertices of an arbitrary edge of T through a simple recursive algorithm. Thus, the proof leads to the following corollary:

Corollary 1. *Given a rooted tree T with $n\,(\geq 2)$ vertices, we can find a Hamiltonian path in T^3 from the root vertex to an arbitrary descendant of the root vertex in $O(n)$ time.*

The function `Hamiltonian_Path_in_TREE_CUBE(`T`,` v`,` w`)` in Figure 2 describes an algorithmic proof of this corollary, which recursively divides a given problem by breaking the connection between the vertex w and its parent. (Throughout this paper, $T_{(u)}$ for a rooted tree T and a vertex u of T denotes the subtree of T rooted at u.) Notice that, as discussed in the proof of Theorem 2, a careless representation of the graph with a conventional adjacency list structure would easily lead to an $O(n^2)$-time algorithm because the elementary operation of deleting an edge from the graph would require $O(n)$ time. Therefore, we also adopt a variant of the adjacency list structure based on doubly-link lists similar to one proposed in the proof, which allows an edge-deletion operation in constant time (please refer to the proof of Theorem 2 for details [24]).

Algorithm 1: Finding a Hamiltonian path in the cube of a tree

```
1  Function Hamiltonian_Path_in_TREE_CUBE($T$, $v$, $w$)
       input  : A tree $T$ rooted at $v$ and a vertex $w$ s.t. $w \neq v$ if
                $|V(T)| > 1$;
       output: A Hamiltonian path $P$ in $T^3$ joining $v$ and $w$;
2      if $V(T) = \{v\}$ then  $P \leftarrow \langle v \rangle$;  return;
       /* Now $w \neq v$, and deleting the edge $(w, \texttt{parent}[w])$ from
          $T$ results in two subtrees, $T \setminus T_{(w)}$ and $T_{(w)}$.        */
3      if parent[$w$] $\neq v$ then $x \leftarrow$ parent[$w$];
4      else if $V(T \setminus T_{(w)}) = \{v\}$ then $x \leftarrow v$;
5      else $x \leftarrow$ a child of $v$ in $T \setminus T_{(w)}$;
6      $P_v \leftarrow$ Hamiltonian_Path_in_TREE_CUBE($T \setminus T_{(w)}$, $v$, $x$);
7      if $V(T_{(w)}) = \{w\}$ then $y \leftarrow w$;
8      else $y \leftarrow$ a child of $w$ in $T_{(w)}$;
9      $P_w \leftarrow$ Hamiltonian_Path_in_TREE_CUBE($T_{(w)}$, $w$, $y$);
10     $P \leftarrow$ the concatenation of $P_v$ and $P_w$ via the edge $(x, y)$ of $T^3$;
11 end
```

Figure 2: The algorithm for finding a Hamiltonian path in the cube of a tree

While Algorithm 1 is a direct consequence of the proof of Theorem 2, we present it here because finding a Hamiltonian path in the cube of a rooted tree between the root vertex and an arbitrary vertex (other than the root vertex if the tree has two or more vertices) plays a key role in finding a paired 2-DPC in the cube of a connected graph.

3. Finding a paired 2-DPC in the cube of a tree

In this section, we first present a linear-time algorithm for finding a paired 2-DPC in the cube of a tree. Then, in the next section, it is combined with the linear-time algorithm for finding a Hamiltonian path in the cube of a tree to build an algorithm for finding a paired 2-DPC in the cube of a connected graph. Although every tree we are dealing with hereafter in this paper is a rooted tree, the *degree* of a vertex refers not to the number of its children but to the number of its neighbors (children and/or parent). Notice that every edge of a tree is a bridge; an edge incident with a leaf node or incident with the root vertex having a single child is trivial, and all the other edges are nontrivial.

3.1. Iterative reduction of a paired 2-DPC problem

We have a paired 2-DPC problem (T, A, B), in which, for a given tree T and two terminal sets $A = \{\alpha_1, \alpha_2\}$ and $B = \{\beta_1, \beta_2\}$ made of four distinct vertices, a paired 2-DPC of T^3 joining A and B is to be found. For the time being, we only consider a *feasible* problem for which a paired 2-DPC exists, implying that A and B satisfy the two conditions of Theorem 1 in T. The fundamental idea of our algorithm for finding a desired 2-DPC is to repeatedly reduce the problem with respect to the graph size until we arrive at a *simple* problem that allows direct construction of a proper solution. That is, given $T_0 = T$, $A_0 = A$, and $B_0 = B$, our algorithm iteratively derives a feasible problem $(T_{k+1}, A_{k+1}, B_{k+1})$ from a feasible one (T_k, A_k, B_k) for $k \geq 0$ such that (i) $|V(T_k)| > |V(T_{k+1})| \geq 4$, (ii) exactly one terminal γ_k in $A_k \cup B_k \overset{\text{def}}{=} \{\alpha_{1_k}, \alpha_{2_k}, \beta_{1_k}, \beta_{2_k}\}$ is replaced with a new terminal γ_{k+1} for the resulting problem so that $(A_k \cup B_k) \setminus \gamma_k = (A_{k+1} \cup B_{k+1}) \setminus \gamma_{k+1}$, and (iii) a solution for (T_k, A_k, B_k) can be *trivially* constructed from that for $(T_{k+1}, A_{k+1}, B_{k+1})$. When the reduction process stops, we directly construct a paired 2-DPC from the last problem, and then repeatedly expand it by following the reduction chain in reverse order, from which a paired 2-DPC of T^3 joining A and B is completed.

In our algorithm, we assume that the input tree T_0 is rooted at some terminal vertex (otherwise, it can be transformed to such a tree in linear time with respect to the tree size), and so are the succeeding trees T_k. Given a k-th feasible problem (T_k, A_k, B_k), the reduction step (i) chooses a vertex p of the tree, called a *pivot vertex*, and a terminal vertex γ_k accordingly, (ii) finds a path spanning all vertices of some partial tree, defined with respect to p and γ_k, (iii) replacing the partial tree with a new terminal vertex γ_{k+1}, and (iv) updates the terminal sets, thus resulting in a contracted tree T_{k+1}

along with new terminal sets A_{k+1} and B_{k+1}, where a part of path in the solution for (T_k, A_k, B_k) is compacted into the new terminal vertex γ_{k+1} in $A_{k+1} \cup B_{k+1}$.

In this iteration step, a key process is which vertex to select from T_k for effective computation. In our algorithm, the pivot vertex is chosen by examining the following three cases in the increasing order of $j = 1, 2, 3$ until a proper one is found:

- **Case j:** T_k has a pivot vertex p that is the parent of a terminal vertex $\gamma_k \in A_k \cup B_k$ such that there exist j terminal vertices that are proper descendants of p in T_k.

Notice that p itself could also be a terminal vertex. When there are multiple candidates for p in Case 1, the higher priority is given to a nonterminal vertex if any. Before delving into details for our algorithm in this section, we should again emphasize that T_k is rooted at a terminal vertex, and the specific algorithm is based on the linear-time Hamiltonian path construction algorithm described in Subsection 2.2.

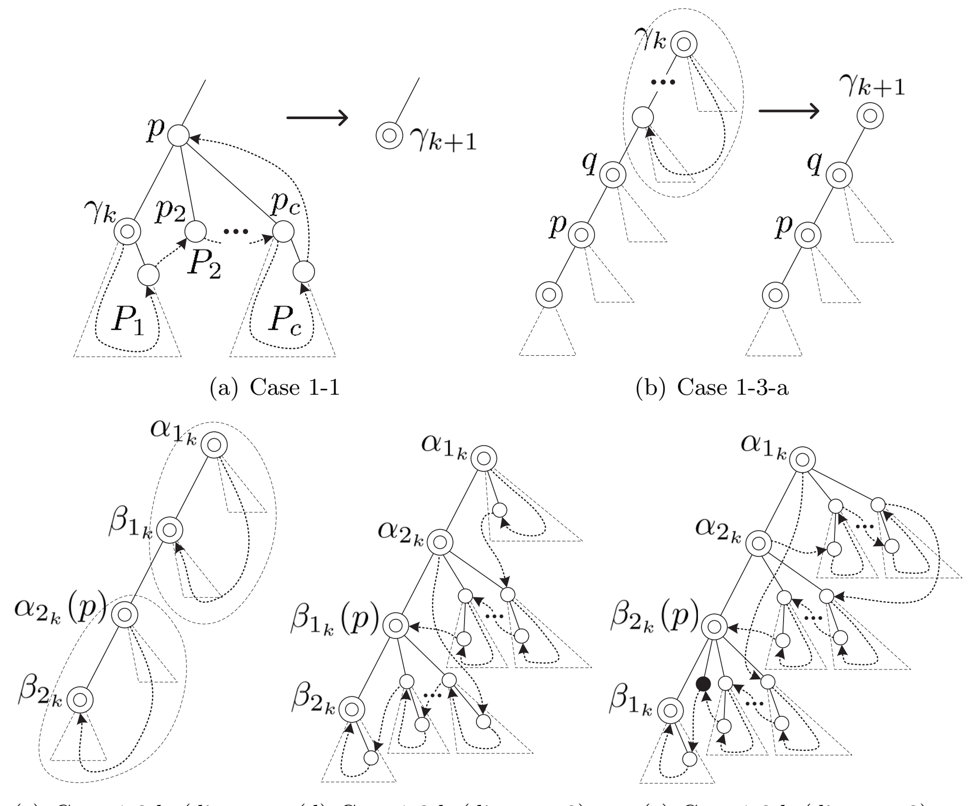
3.2. Manipulation of Case 1

The first case is further divided into the following three subcases according to the kinds of the pivot vertex p and its parent:

- **Case 1-1:** p is a nonterminal vertex.
- **Case 1-2:** p is a terminal vertex, but the parent of p is a nonterminal vertex.
- **Case 1-3:** Both p and its parent are terminal vertices.

3.2.1. Details for Case 1-1 and Case 1-2

Figure 3(a) illustrates an example situation of Case 1-1, where the pivot vertex p has c children $p_l, l = 1, 2, \ldots, c\,(c \geq 1)$, and, without loss of generality, p_1 is assumed to be the terminal vertex γ_k. Then, for each subtree $T_{k(p_l)}$ (recall that $T_{k(p_l)}$ is the subtree of T_k rooted at p_l), our algorithm finds a Hamiltonian path P_l of $T^3_{k(p_l)}$ that starts from p_l and ends at some child p'_l of p_l (if $|V(T_{k(p_l)})| \geq 2$) or a one-vertex path $P_l = \langle p_l \rangle$ (if $|V(T_{k(p_l)})| = 1$). Then, it is clear to see that $P_1 \circ P_2 \circ \cdots \circ P_c \circ \langle p \rangle$, the concatenation of the paths extended to p, becomes a legitimate Hamiltonian path of $T^3_{k(p)}$ from $\gamma_k\,(= p_1)$ to p since the distance from the last vertex of P_c and p in T_k is at most 2. Once the Hamiltonian path is constructed, all the subtrees $T_{k(p_l)}$ are



(a) Case 1-1

(b) Case 1-3-a

(c) Case 1-3-b (distance 1)

(d) Case 1-3-b (distance 2)

(e) Case 1-3-b (distance 3)

Figure 3: Illustrations for Case 1

eliminated from T_k, and p is replaced by a new terminal γ_{k+1} that functions the same as γ_k and remembers the found Hamiltonian path from γ_k to p.

Once this iteration step is over, we end up in a new problem $(T_{k+1}, A_{k+1}, B_{k+1})$ with a contracted tree and updated terminal sets. Note that a solution for (T_k, A_k, B_k) can easily be constructed from a paired 2-DPC for $(T_{k+1}, A_{k+1}, B_{k+1})$ simply by replacing γ_{k+1} in the path cover with the Hamiltonian path stored in γ_{k+1}. Also, the new terminal γ_{k+1} is a leaf node in T_{k+1}. Thus, the *feasibility* of the problem $(T_{k+1}, A_{k+1}, B_{k+1})$ is guaranteed since γ_{k+1} may even not be in a neighborhood of a pure bridge vertex in T_{k+1} (recall the conditions of Theorem 1 again).

In addition, the parent of p can also be reached in T_k^3 from the last vertex of the Hamiltonian path of the last subtree $T_{k(p_c)}$. Thus, the same iteration step can be applied to Case 1-2 except that the parent of p, instead of p, is replaced with γ_{k+1}. Again, the feasibility of the paired 2-DPC problem is preserved since the terminal vertex p now becomes a leaf node in T_{k+1}.

3.2.2. Details for Case 1-3

The situation in Case 1-3 is somewhat more complicated than those in the previous two subcases. Our algorithm subdivides this subcase further as follows, in which the grandparent of the pivot vertex p is additionally examined:

- **Case 1-3-a:** The grandparent of p is a nonterminal vertex.
- **Case 1-3-b:** The grandparent of p is a terminal vertex, which should then be the root vertex.
- **Case 1-3-c:** p has no grandparent, implying that its parent is the root vertex.

When both p and the parent of p are terminal vertices, but the grandparent of p is not as in Case 1-3-a, the reduction process is simple (refer to Figure 3(b)). Let q be the parent of p and, instead of the original γ_k which is a child of p, choose the root vertex as γ_k. Also, consider the tree $T_k \setminus T_{k(q)}$, induced by $V(T_k) \setminus V(T_{k(q)})$, which has at least two vertices. Then, our algorithm finds a Hamiltonian path in the cube of this partial tree connecting γ_k and the parent of q, and replaces the tree with a new terminal vertex γ_{k+1}, storing the found Hamiltonian path with it. Again, the degree of γ_{k+1} is one in T_{k+1}, thus the feasibility keeps effective in the reduced problem.

Now, consider Case 1-3-b. Then, the four terminal vertices line up in the tree along a path from the root vertex where, among the 24 (= 4!) possible orders of them, three distinct classes of combinations exist as illustrated in Figures 3(c) to 3(e). (Without loss of generality, assume that α_{1_k} is the root vertex. Then, the 24 combinations can be classified according to the distance in the tree between α_{1_k} and β_{1_k}, which may be 1, 2, or 3.) In either case, the current problem (T_k, A_k, B_k) is not reduced to a smaller problem, but its solution is constructed directly.

First, when the distance between them is one as shown in Figure 3(c), we consider the two partitioning trees of T_k obtained by disconnecting the edge $(\beta_{1_k}, \alpha_{2_k})$. Then, any Hamiltonian paths in the cubes of the respective partial trees connecting α_{1_k} and β_{1_k}, and α_{2_k} and β_{2_k} constitute an appropriate paired 2-DPC of T_k^3. Second, take a look at the case of distance two shown in Figure 3(d). It is not difficult to see that, similar to Case 1-1, the

Hamiltonian paths in the cubes of $T_{k(\alpha_{1_k})} \setminus T_{k(\alpha_{2_k})}$ and the subtrees, if any, rooted at the children of α_{2_k} except β_{1_k} can be stitched through edges of T_k^3 to form an α_{1_k}–β_{1_k} path in T_k^3. Then, this path constitutes a proper 2-DPC of T_k^3 along with an α_{2_k}–β_{2_k} path that is constructed similarly (recall that each subtree might be a one-vertex tree).

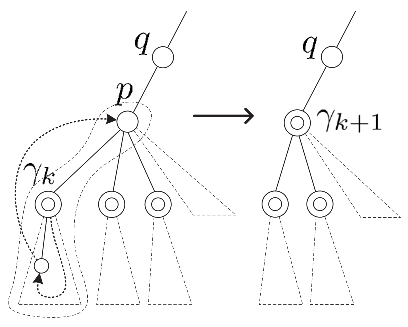
Third, a special treatment is necessary for the case of distance three shown in Figure 3(e). Note that the second condition of Theorem 1 requires that the two vertices α_{2_k} and β_{2_k} should not form a pure bridge pair in T_k, meaning at least one of them is not a pure bridge vertex. When β_{2_k} is not, it has at least one child of degree one, as marked with a black solid circle, because it must be incident to at least one trivial bridge. (Of course, β_{1_k} may be of degree one.) Consider a set of Hamiltonian paths in the cubes of the subtrees rooted at the children of β_{2_k}, in which each Hamiltonian path connects the respective root to its child (if the subtree has two or more vertices) or to itself (if the subtree has a single vertex). Note that at least one of these Hamiltonian paths is a one-vertex path including that rooted at the degree-one child of β_{2_k}. Then, the degree-one child allows to connect these Hamiltonian paths to form an α_{1_k}–β_{1_k} path in T_k^3, and moreover, an α_{2_k}–β_{2_k} path can be constructed similarly. If β_{2_k} is a pure bridge vertex but α_{2_k} is not, we restructure T_k in time linear to $|V(T_k)|$ in such a way that β_{1_k} becomes the root vertex, and proceed in the same way.

Finally, take a look at Case 1-3-c where the three terminal vertices lines up in the tree starting from the root vertex, i.e. the parent of p. Then the fourth terminal vertex must be a child of the root vertex because otherwise the nonterminal parent of the terminal vertex should have been chosen as p as in Case 1-1. In this case, we again restructure T_k in time proportional to $|V(T_k)|$ such that this fourth terminal vertex becomes the root vertex, whose case in turn becomes identical to Case 1-3-b.

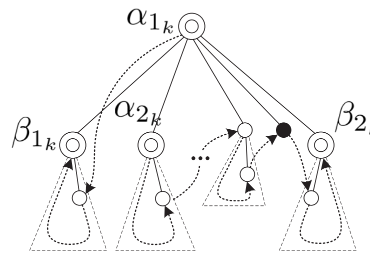
3.3. Manipulation of Case 3

Next, in this subsection, we investigate Case 3 first for a simpler description of our algorithm. When a pivot vertex p has three terminal vertices as proper descendants, these three terminal vertices must be children of p because, otherwise, other vertex would have been chosen as p for Case 1 or Case 2. In Case 3, we consider two major subcases as follows:

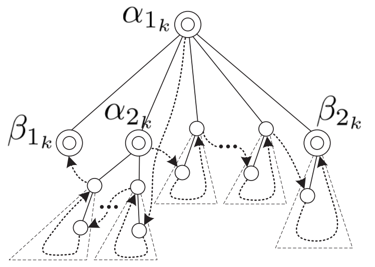
- **Case 3-1:** p is a nonterminal vertex.
- **Case 3-2:** p is a terminal vertex.



(a) Case 3-1



(b) Case 3-2-a



(c) Case 3-2-b

Figure 4: Illustrations for Case 3

When Case 3-1 occurs, a reduction process proceeds as implied in Figure 4(a), where one of the three terminal vertex is selected as γ_k, and a Hamiltonian path in the cube of $T_{k(\gamma_k)}$ from γ_k to a child of γ_k is extended to p (recall that the Hamiltonian path could be a one-vertex path if $T_{k(\gamma_k)}$ has only one vertex). Then, the partial tree made of p and $T_{k(\gamma_k)}$ is contracted into the new terminal vertex γ_{k+1}. A critical factor in this reduction process is to guarantee the feasibility in the resulting problem. In particular, when q, the parent of p, is a terminal vertex, and thus q is the root vertex, the first condition of Theorem 1 would be broken in the new problem if the new terminal vertex γ_{k+1} becomes a pure bridge vertex. Observe that, when q is a terminal vertex, p has at least one adjacent vertex in T_k, whether a child or the parent, whose degree is one because otherwise the four terminals in $A_k \cup B_k$ belong to a closed neighborhood of the pure bridge vertex p in T_k, violating the first condition of Theorem 1 with respect to the current problem (T_k, A_k, B_k). Thus, when choosing γ_k from the three terminal vertices, we choose one, if any, that has a child, which will always guarantee the feasibility in the reduced problem.

When Case 3-2 happens, that is, if p is a terminal vertex, and thus is the root vertex of T_k, a proper solution is constructed directly. Without loss of generality, assume that the root vertex is α_{1_k}. Then, the construction procedure differs according to whether β_{1_k}, the matching terminal vertex, has a child or not.

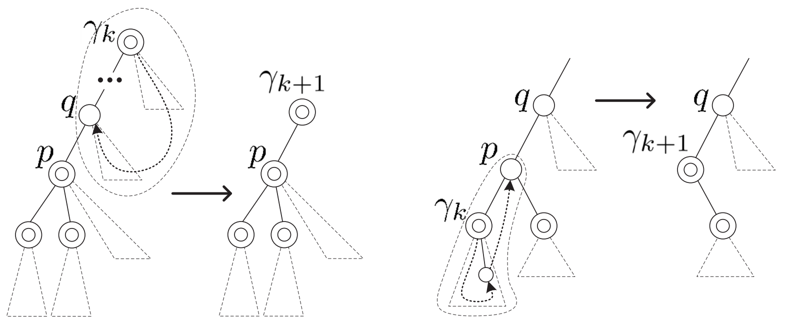
- **Case 3-2-a:** β_{1_k} has a child.
- **Case 3-2-b:** β_{1_k} does not have a child.

In Case 3-2-a when β_{1_k} has a child, the one-vertex path $\langle \alpha_{1_k} \rangle$ is concatenated with a Hamiltonian path in the cube of $T_{k(\beta_{1_k})}$ from a child of β_{1_k} to β_{1_k} to form an α_{1_k}–β_{1_k} path in T_k^3. On the other hand, consider the subtrees rooted at the children of α_{1_k} except β_{1_k}, and the corresponding Hamiltonian paths found in the cubes of those subtrees traversing from the their roots to the respective children, if any. Observe that the root vertex α_{1_k} has as least one child whose degree is one (drawn using a black solid circle in Figure 4(b)) so that it would not be a pure bridge vertex holding the other three terminal vertices in the closed neighborhood in T_k. This means that there exists at least one one-vertex path in the found Hamiltonian paths which allows to stitch the Hamiltonian paths to form an α_{2_k}–β_{2_k} path in T_k^3, completing a paired 2-DPC solution. In Case 3-2-b when β_{1_k} has no child, we consider two groups of subtrees, if any, in T_k (refer to Figure 4(c)). The first one is those rooted at the children of α_{2_k}, and the other one is those rooted at the children of α_{1_k} excluding β_{1_k} and α_{2_k}. A Hamiltonian path is found in the cube of each subtree from a child, if any, of the root vertex to the root vertex. Then, the paths corresponding to the first group are assembled to form an α_{1_k}–β_{1_k} path in T_k^3, while those for the second group to form an α_{2_k}–β_{2_k} path, which is now possible because α_{2_k} has no descendants to take care of.

3.4. Manipulation of Case 2

Finally, when Case 2 is selected, the two terminal vertices counted in the selection process should be children of the pivot vertex p because otherwise Case 1 would have been chosen. We consider the following three subcases:

- **Case 2-1:** p is a terminal vertex, but its parent is not.
- **Case 2-2:** Both p and its parent are terminal vertices.
- **Case 2-3:** p is a nonterminal vertex.



(a) Case 2-1 (b) Case 2-3

Figure 5: Illustrations for Case 2

First in Case 2-1, the root vertex is chosen as γ_k, and a reduction process proceeds where a Hamiltonian path in the cube of $T_k \setminus T_{k(p)}$ from γ_k to q, the parent of p, is contracted to the new terminal vertex γ_{k+1} (see Figure 5(a)). If both p and q are terminal vertices as in Case 2-2, it means that q is the root vertex. Then, we transform T_k in linear time such that the new tree is rooted at p that now has three terminal vertices as children, in which case a 2-DPC solution is constructed directly as explained in Case 3-2. For the last case of Case 2-3, one of the two children of p is selected as γ_k, and a simple reduction process is performed where a Hamiltonian path in the cube of $T_{k(\gamma_k)}$ from γ_k to one of its children, which is then extended to p, is contracted to the new terminal vertex γ_{k+1} (see Figure 5(b)). In this simple process, we may have to be careful in selecting γ_k because, if both q and its neighbor (child or parent) other than p are terminal vertices, the four terminal vertices will line up after the reduction process. Thus, in order to guarantee not to break the second condition of Theorem 1 in the new problem, we choose one among the two children of p that does not match q if q happens to be a terminal vertex (for example, if q is α_{1_k}, one that is not β_{1_k} is selected as γ_k).

3.5. Algorithm and time complexity

The algorithm described in this section is summarized in Figure 6. The crucial part of our algorithm is to determine what action to take for the current problem (T_k, A_k, B_k) among the choices of the reduction (Case 1-1, Case 1-2, Case 1-3-a, Case 2-1, Case 2-3, and Case 3-1) and the direct construction (Case 1-3-b, Case 1-3-c, Case 2-2, and Case 3-2), where the decision can easily be made in constant time once the pivot vertex is located. For efficient selection of the pivot vertex during the repeated reduction processes, we store in each vertex v of the tree a value `num_terms[v]` that holds the number of terminal vertices that are proper descendants of v in T_k. Given the input tree T_0, `num_terms[v]` can be initialized in time linear to the size of the tree, i.e. $|V(T_0)|$, through a simple depth first search (Line 2), during which (i) when a vertex v is first visited, `num_terms[v]` is set to one if v is a terminal vertex, or zero otherwise, and (ii) when the search for the subtree rooted at v is finished, `num_terms[v]` is added to that of its parent. Then, the `num_terms[v]` array is completed by decreasing the value by 1 for each of the four terminals.

Algorithm 2: Finding a paired 2-DPC in the cube of a tree

```
 1 Function Paired_2-DPC_in_TREE_CUBE($T$, $A$, $B$)
       input  : A rooted tree $T$ and terminal sets $A = \{\alpha_1, \alpha_2\}$ and
                $B = \{\beta_1, \beta_2\}$ that satisfy the two conditions of
                Theorem 1;
       output: A paired 2-DPC $\{P_1, P_2\}$ of $T^3$ joining $A$ and $B$;
 2     Initialize the num_terms[v] array;
 3     $T_0 \leftarrow T$, $A_0 \leftarrow A$, $B_0 \leftarrow B$;
 4     $k = 0$;
 5     repeat
 6         Select a pivot vertex $p$ from $(T_k, A_k, B_k)$;
 7         If a reduction is possible, reduce $(T_k, A_k, B_k)$ into
           $(T_{k+1}, A_{k+1}, B_{k+1})$;
 8         $k \leftarrow k + 1$;
 9     until a reduction is impossible;
10     Find a paired 2-DPC $\{P_{1_k}, P_{2_k}\}$ for $(T_k, A_k, B_k)$ via a direct
       construction;
       /* Now, one of the paths $P_{1_k}$ and $P_{2_k}$ contains $\gamma_k$.     */
11     repeat
12         Expand $\{P_{1_k}, P_{2_k}\}$ into $\{P_{1_{k-1}}, P_{2_{k-1}}\}$ by replacing $\gamma_k$ with
           the Hamiltonian path containing $\gamma_{k-1}$;
13         $k \leftarrow k - 1$;
14     until $k > 0$;
15     $\{P_1, P_2\} \leftarrow \{P_{1_0}, P_{2_0}\}$;
16 end
```

Figure 6: The algorithm for finding a paired 2-DPC in the cube of a tree

Observation 1. *Given a tree T with n vertices, the* `num_terms[v]` *array can be initialized in time proportional to n.*

Now, consider the process of reducing the current problem (T_k, A_k, B_k) to the new problem $(T_{k+1}, A_{k+1}, B_{k+1})$ in Lines 6 to 8. First, the pivot vertex can be selected in constant time by comparing the `num_terms[v]` values of the parents of the three terminal vertices not being the root vertex. Furthermore, if we let n_k and n_{k+1} be $|V(T_k)|$ and $|V(T_{k+1})|$, respectively, the partial tree that will be compressed to a single terminal vertex will have $n_k - n_{k+1} + 1$ vertices, which can be identified in constant time. Then, by Corollary 1, it is not difficult to see that the Hamiltonian path (Case 1-3-a and Case 2-1), the extended Hamiltonian path (Case 2-3 and Case 3-1), and the extended sequence of Hamiltonian paths (Case 1-1 and Case 1-2) can all be found in time linear to $n_k - n_{k+1}$. Once the new contracted tree is produced, the `num_terms[v]` must be updated for the next iteration. However, this computation is just simple because only the value of the new terminal vertex except Case 1-2 where that of the pivot vertex p also needs to be updated can be set to the new value properly in constant time. Adding all the costs in the single reduction step, we arrive at the following fact:

Observation 2. *When a reduction occurs for the tree T_k with n_k vertices, the computation time $T(n_k)$ for T_k can be described in terms of that for the new tree T_{k+1} with n_{k+1} vertices as $T(n_k) \leq T(n_{k+1}) + c_{rd} \cdot (n_k - n_{k+1} + 1)$ for some constant c_{rd}.*

Finally, consider the cost for the direct construction of a paired 2-DPC performed in Line 10. First of all, the two partial trees that partition T_k, for which the two disjoint paths will cover respectively, can be identified in time linear to $|V(T_k)|$ (Case 1-3-b, Case 1-3-c, Case 2-2, and Case 3-2). (Notice that it is enough to enumerate for each of the two disjoint covering paths the root vertices of the subtrees for which Hamiltonian paths are to be found, and their numbers are bounded by $|V(T_k)|$.) Furthermore, in any case, we can see that the total cost for finding the Hamiltonian paths and connecting them is also bounded by $O(|V(T_k)|)$.

Observation 3. *When a direct construction occurs for a tree T_k with n_k vertices, the computation time is $T(n_k) \leq c_{dc} \cdot n_k$ for some constant c_{dc}.*

Theorem 3. *Given a tree T with $n\,(\geq 4)$ vertices and two terminal sets $A = \{\alpha_1, \alpha_2\}$ and $B = \{\beta_1, \beta_2\}$ that satisfy the two conditions of Theorem 1, we can find a paired 2-DPC of T^3 joining A and B in $O(n)$ time.*

Proof. Assume that, for the given input tree $T\,(= T_0)$ with $n\,(= n_0)$ vertices, the reduction stops at T_k with n_k vertices, for which a paired 2-DPC is constructed directly. Then, by applying the recursive formula in Observation 2 repeatedly, and the cost function in Observation 3 lastly, we are led to

$$\begin{aligned}T(n) &\le T(n_k) + c_{rd} \cdot \textstyle\sum_{i=0}^{k-1}(n_i - n_{i+1} + 1)\\ &\le c_{dc} \cdot n_k + c_{rd} \cdot (n_0 - n_k + k)\\ &\le \max\{c_{dc}, c_{rd}\} \cdot (n_0 + k) \le 2 \cdot \max\{c_{dc}, c_{rd}\} \cdot n = O(n).\end{aligned}$$

Constructing the final solution $\{P_1, P_2\}$ by the simple backward expansions of the compressed paths in Lines 12 and 13 involves at most n vertices in total. Thus, the entire construction computation including the linear-time initialization, described in Observation 1, can be done in time proportional to the number of vertices of the input tree. $\square$

4. Finding a paired 2-DPC in the cube of a connected graph

Now, we are ready to describe a linear-time algorithm for finding a paired 2-DPC of G^3 for a given connected graph G and two terminal sets $A = \{\alpha_1, \alpha_2\}$ and $B = \{\beta_1, \beta_2\}$.

4.1. Finding a solution via a spanning tree possibly augmented with an edge

The key idea of our main algorithm is to transform the problem for an arbitrary graph into a problem for a spanning tree, which provides a simpler solution. We first show in the next theorem that whenever a paired 2-DPC joining two terminal sets exists in the cube of a graph G, there is a spanning tree T of the graph such that the cube of the spanning tree augmented with at most one edge e has a paired 2-DPC joining the same terminal sets. Be noticed that a necessary condition for two terminal sets to break the conditions of Theorem 1 for a connected graph is either the four terminal vertices are in the closed neighborhood of some vertex (the C1 condition), or they form a path of length three in the graph in such a way that matching pairs of terminal vertices have the same degrees in the path (the C2 condition). In the proof of the theorem, we shall call two terminal sets *strongly admissible* for a graph if they do not even meet this minimum requirement with respect to the graph, guaranteeing the existence of a paired 2-DPC in the cube of the graph joining the two terminal sets.

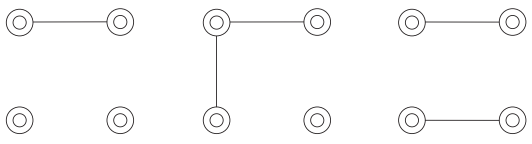
Theorem 4. *Given a connected graph G, let two terminal sets $A = \{\alpha_1, \alpha_2\}$ and $B = \{\beta_1, \beta_2\}$ satisfy the two conditions, $C1$ and $C2$, of Theorem 1. Then, G always has a spanning tree T such that T^3 or $(T + e)^3$ for some edge $e \in E(G) \setminus E(T)$ has a paired 2-DPC joining A and B.*

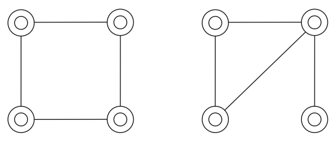
Proof. If there exists a terminal vertex $\gamma \in A \cup B$ that is not a cut vertex in G, imagine a depth-first-search tree (a DFS tree in short) T of G rooted at γ. Then, T meets the conditions of Theorem 1 with respect to A and B, implying the existence of a paired 2-DPC of T^3 joining A and B because γ, which has only one child, is connected to T only through a trivial bridge. Therefore, from now on in this proof, we shall assume that every terminal vertex is a cut vertex in G.

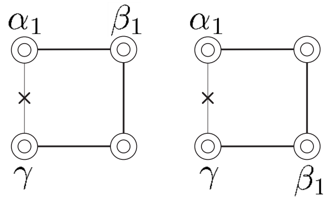
The remaining proof proceeds by classifying the given graph G in terms of the number of edges in the subgraph H that is induced by the four terminal vertices in $A \cup B$. (For the sake of easier understanding, we shall investigate four cases in the order of increasing complexity.) Note that if T is a spanning tree such that $|E(H) \cap E(T)| \geq 1$ as in the first three cases, the closed neighborhood of a nonterminal vertex in T cannot include all the four terminal vertices to break the C1 condition of Theorem 1.

Case 1: $|E(H)| = 1$ or 2. In this case, H is in the form of one of the three graphs in Figure 7(a) that illustrates how four terminal vertices may be connected through one or two edges. Then, for any terminal vertex γ of degree one in H, consider a DFS tree T of G that is rooted at γ and contains all the edges of H. (Such a DFS tree can easily be built by putting the edges of H in the front of the respective lists in the adjacency list representation of G.) Then, it is obvious that A and B are strongly admissible for the spanning tree T, and therefore T^3 has a paired 2-DPC joining A and B.

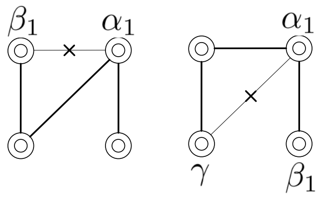
Case 2: $|E(H)| \geq 4$. In this case, consider a graph H_4 with four edges that results from removing arbitrary $|E(H)| - 4$ edges from H. Then, there are two kinds of H_4s as illustrated in Figure 7(b). When H_4 is a cycle of length four (see Figure 7(c)), choose a terminal vertex γ that is not β_1 but is adjacent to α_1. Then, for a spanning tree T of G that is rooted at α_1 and contains all the edges of H_4 except (α_1, γ), A and B are strongly admissible for T, and therefore we have a desired 2-DPC. When H_4 has a cycle of length three (see Figure 7(d)), assume without loss of generality that α_1 is the terminal vertex of degree three in H_4. Imagine a path of length three in H_4 that results from deleting an edge (α_1, γ), where γ is β_1 if β_1 has degree two in H_4 or a terminal vertex other than α_1 and β_1 otherwise. Then, any spanning tree T of G that is rooted at β_1 and contains all the tree edges


(a) Possible Hs in Case 1


(b) Possible H_4s in Case 2



(c) Edge deletion in H_4 (cycle of length 4)

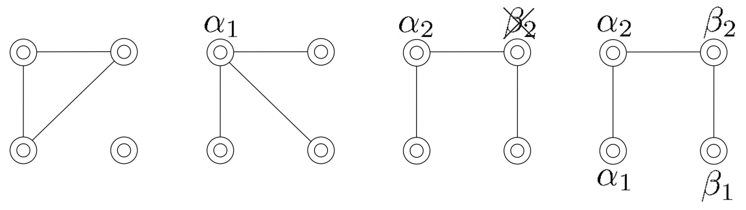


(d) Edge deletion in H_4 (cycle of length 3)

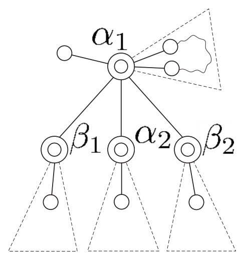
Figure 7: Illustrations for Cases 1 and 2 of Theorem 4

of the path will make A and B strongly admissible with respect to itself, proving the theorem.

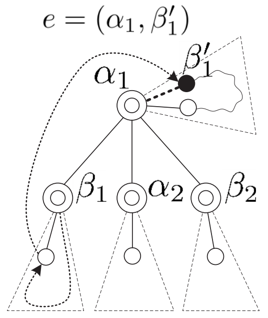
Case 3: $|E(H)| = 3$. There are three different kinds of Hs depending on how the three edges connect the four terminal vertices. First, if H is made of an isolated vertex and a cycle of length three as shown in the first graph in Figure 8(a) (**Case 3-1**), it is enough to delete from G an arbitrary edge of the length-three cycle, and construct a DFS tree T of G that is rooted at any terminal vertex incident to the deleted edge and contains the remaining two edges of H. Clearly, A and B are strongly admissible for the spanning tree T, whose cube then has a paired 2-DPC joining them. Second, let H be a 3-leaf star tree whose center vertex is assumed to be α_1 without loss of generality (see the second graph in Figure 8(a)). If there exists an edge of H that is not a bridge in G, it suffices to remove the edge from G, still allowing G to be connected, and find a spanning tree T of G that is rooted at a terminal vertex not incident to the deleted edge and contains the remaining two edges of H, which would make A and B strongly admissible for T. If all the three edges of H are bridges in G (note that they are *nontrivial* because the terminal vertices are all cut vertices as assumed in the proof), there must exist an edge, a trivial bridge or a non-bridge, incident to α_1 that would prevent α_1 from ever being a pure bridge vertex in G, because otherwise A and B would break the condition $C1$ of Theorem 1 with respect to G. That is, α_1 must be incident to a trivial bridge and/or must be adjacent to some two other vertices such that the three vertices are contained in a cycle (see Figure 8(b)).



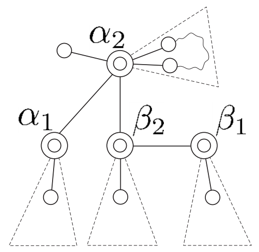
(a) Possible Hs



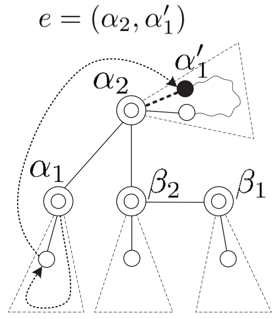
(b) Second subcase where α_1 is not a pure bridge vertex



(c) Hybrid construction for the case of (b)



(d) Fourth subcase where α_2 is not a pure bridge vertex



(e) Hybrid construction for the case of (d)

Figure 8: Illustrations for Case 3 of Theorem 4

If there exists a trivial bridge incident to α_1 (**Case 3-2-a**), it is sufficient to find a DFS tree T of G rooted at α_1 in which α_1 cannot be a pure bridge vertex, allowing T^3 to have a paired 2-DPC joining A and B. If there is no such trivial bridge, but is a cycle containing α_1 (**Case 3-2-b**), construct a spanning tree T of G by performing a depth-first search starting from α_1. Then, one of the two edges in the cycle that are incident to α_1 must turn out to be a back edge in the search. If we denote the other end vertex of this edge by β_1' (see Figure 8(c)), then it is clear that A and $\{\beta_1', \beta_2\}$ are strongly admissible with respect to the tree $T \setminus T_{(\beta_1)}$ (recall that $T_{(\beta_1)}$ is the subtree of T rooted at β_1), implying that there exists a paired 2-DPC of the cube of $T \setminus T_{(\beta_1)}$, made of an α_1–β_1' path and an α_2–β_2 path. Now, imagine a Hamiltonian path of $T^3_{(\beta_1)}$ that connects β_1 to a child of β_1. If we add an edge $e = (\alpha_1, \beta_1')$ to the spanning tree T, this Hamiltonian path can be combined with the α_1–β_1' path in the graph $(T+e)^3$ through the child vertex to β_1', forming an α_1–β_1 path. Together with the α_2–β_2, the concatenated path then constitutes a paired 2-DPC of $(T+e)^3$ joining A and B, proving the existence of the spanning tree T and an edge e of the theorem.

For the two remaining cases of Hs in which H is a path of length three, assume without loss of generality that α_2 is an internal vertex. If the other internal vertex is not β_2 as shown in the third graph in Figure 8(a) (**Case 3-3-a**), A and B will be strongly admissible for a spanning tree T of G that is rooted at a vertex of degree one in H and contains all the three edges of H, proving the theorem. If the other internal vertex is β_2 (the fourth one in Figure 8(a)), at least one of α_2 and β_2 must be a non-pure bridge vertex so that A and B satisfy the conditions of Theorem 1. If we assume without loss of generality that α_2 is not a pure bridge vertex, α_2 must be incident to a trivial bridge (**Case 3-3-b-i**) and/or must be contained in a cycle (**Case 3-3-b-ii**). Then, the existence of T or $T + e$ that the theorem says can similarly be found as before except that α_2 and α_1 now behave as α_1 and β_1 of the second subcase, respectively (refer to Figures 8(d) and 8(e)), completing the proof for Case 3.

Case 4: $|E(H)| = 0$. In this case, there exist no edges between the four terminal vertices, and therefore only the first condition $C1$ of Theorem 1, requiring that the four terminal vertices should not be in the closed neighborhood of some pure bridge vertex, might be broken for a spanning tree of G. First, if there is no (non-terminal) vertex v in G, whether it is a pure bridge vertex or not, such that $A \cup B \subseteq N_G[v]$ (**Case 4-1**), then it is obvious that A and B are strongly admissible for any spanning tree T of G. Second, if there is more than one such vertex (**Case 4-2**), α_1 will be at least distance-three away from some other terminal vertex in any DFS tree T of G rooted at α_1, implying that A and B are again strongly admissible for T. Third, assume for the rest of the proof that there exists a unique (non-terminal) vertex v such that $A \cup B \subseteq N_G[v]$. If there is a terminal vertex, say α_1 such that (v, α_1) is not a bridge (**Case 4-3-a**), we can build a spanning tree T of G from $G \setminus (v, \alpha_1)$ rooted at v for which A and B are always strongly admissible. If all the four edges from v to the terminal vertices are bridges in G (again they must be nontrivial), the proof proceeds very similar to the second subcase of Case 3. That is, in order for the first condition $C1$ of Theorem 1 to be met for G, v should be incident to a trivial bridge and/or should be adjacent to some two other vertices such that the three vertices are contained in a cycle. If a trivial bridge is adjacent to v (**Case 4-3-b-i**), finding a DFS tree T of G rooted v will suffice to prove the theorem.

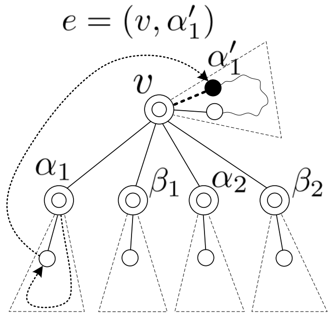


Figure 9: Illustrations for Case 4 of Theorem 4

If there is no such trivial bridge, but is a cycle containing v (**Case 4-3-b-ii**), a depth-first search is carried out starting from v to find a spanning tree T of G, where we call α_1' the vertex in the cycle that, together with v, forms a back edge in the search (refer to Figure 9). Then, $\{\alpha_1', \alpha_2\}$ and B are strongly admissible for $T \setminus T_{(\alpha_1)}$, guaranteeing the existence of a paired 2-DPC in the cube of $T \setminus T_{(\alpha_1)}$, made of an α_1'–β_1 path and an α_2–β_2 path. Similar to before, if we add an edge $e = (v, \alpha_1')$ to the spanning tree T, a Hamiltonian path of $T^3_{(\alpha_1)}$ from α_1 to a child of α_1 can be connected to the α_1'–β_1 path of the paired 2-DPC in $(T + e)^3$ through the edge from the child of α_1 to α_1'. This implies the existence of a spanning T and an edge e for which $(T + e)^3$ has a paired 2-DPC joining A and B, completing the entire proof of the theorem. □

Remark 1. Given a connected graph G whose cube has a paired 2-DPC joining two terminal sets A and B, there often exists a spanning tree T of G whose cube also has a paired 2-DPC joining the same terminal sets.

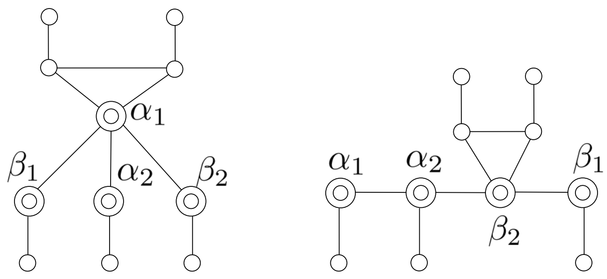


Figure 10: Any spanning tree of the graph shown in the left would violate the condition *C1* of Theorem 1. On the other hand, any spanning tree of the graph in the right would violate the condition *C2*.

However, this is not always true as illustrated in Figure 10, in which a unicyclic subgraph of a connected graph is considered for our algorithm to break one of the two conditions of Theorem 1 and thus to be able to build a desired paired 2-DPC.

4.2. Algorithm and time complexity

The proof of Theorem 4 is constructive in nature, and is easily led to an efficient procedure. Our main algorithm is summarized in Algorithm 3, which consists of four major stages. In the first step (Lines 2 and 3), a pre-computation is performed to decide if the edges of the graph G are bridges and if the terminal vertices in A and B are cut vertices. Once obtained, this information is used effectively to answer each question raised in the following three stages. Before starting to find a paired 2-DPC of G^3, the algorithm checks in the second step if a solution ever exists (Line 4), and continues only if it does. In the third (Lines 6 to 9) and the fourth (Lines 10 to 24) steps, which are an algorithmic description of the proof of Theorem 4, the paired 2-DPC is actually computed. First, if there is a terminal vertex v_R that is not a cut vertex, we use Algorithm 2 (`Paired_2-DPC_in_TREE_CUBE`) to find a paired 2-DPC of T^3 joining A and B for a spanning tree T of G rooted at v_R, which becomes the desired solution for G^3. If all the four terminal vertices are cut vertices, the fourth step, which is actually the main part of the proof, is performed.

Observe that each subcase occurred in the proof of Theorem 4 can uniquely be mapped into one of two types of operations, *type-A* or *type-B* (refer to the caption of Figure 11). For the type-A operation (Lines 12 and 16), a depth-first search is done to build a spanning tree T of G which

Algorithm 3: Construction of a paired 2-DPC in the cube of a graph

```
1  Function Paired_2-DPC_in_GRAPH_CUBE($G$, $A$, $B$)
       input  : A connected graph $G$ and terminal sets $A = \{\alpha_1, \alpha_2\}$
                and $B = \{\beta_1, \beta_2\}$;
       output: A paired 2-DPC $\{P_1, P_2\}$ of $G^3$ joining $A$ and $B$;
2      Determine if each edge is a bridge in $G$;
3      Determine if each terminal vertex is a cut vertex in $G$;
4      Determine if both two conditions of Theorem 1 are satisfied;
5      if not satisfied then  $\{P_1, P_2\} \leftarrow \{\textbf{null}, \textbf{null}\}$;  return;
6      if a terminal vertex $v_R$ is not a cut vertex then
7          $(T, \textbf{null}) \leftarrow$ DFS($G$, $v_R$, null, null, 0);
8          $\{P_1, P_2\} \leftarrow$ Paired_2-DPC_in_TREE_CUBE($T$, $A$, $B$);
9          return;
10     Classify the current case according to $H = G[A \cup B]$;
11     if it is type-A then
12         Pick the vertex $v_R$ from which a spanning tree $T$ is built;
13         Set the set of edges $X$ to be excluded in $T$;
14         Set the set of edges $I$ to be included in $T$;
15         $(T, \textbf{null}) \leftarrow$ DFS($G$, $v_R$, $X$, $I$, 0);
16         $\{P_1, P_2\} \leftarrow$ Paired_2-DPC_in_TREE_CUBE($T$, $A$, $B$);
17     else // type-B
18         Pick the vertex $v_R$ from which a spanning tree $T$ is built;
19         $(T, u) \leftarrow$ DFS($G$, $v_R$, null, null, 1);
           /* Now, the edge $e = (v_R, u)$ is a back edge of $T$.   */
20         Pick the terminal $w$ s.t. a Hamiltonian path is sought in $T^3_{(w)}$;
21         $P_{11} \leftarrow$ Hamiltonian_Path_in_TREE_CUBE($T_{(w)}$, $w$, child($w$));
22         $(A', B') \leftarrow (A, B) - w + u$;  // Replace $w$ by $u$ in $A$ & $B$.
23         $\{P_{12}, P_2\} \leftarrow$ Paired_2-DPC_in_TREE_CUBE($T \setminus T_{(w)}$, $A'$, $B'$);
24         $P_1 \leftarrow$ CombinePaths($P_{11}$, $P_{12}$, $u$, child($w$));
25 end
```

Figure 11: The algorithm for finding a paired 2-DPC in the cube of a graph. In Line 10, the current case is classified as one of the 13 different subcases specified in the proof of Theorem 4, where Case 3-2-b, Case 3-3-b-ii, and Case 4-3-b-ii belong to *type-B*, and the others to *type-A*. A call to function `DFS(`G`, `v_R`, `X`, `I`, `f`)` for a connected graph G, a vertex v_R, two edge sets X and I, and a Boolean flag f returns a pair (T, u), where T is a spanning tree of G with root v_R s.t. $X \cap E(T) = \emptyset$ and $I \subseteq E(T)$, and u is a vertex s.t. $e = (v_R, u)$ is a back edge of T if f is 1, or undefined otherwise. Also, the function `CombinePaths(`P_{11}`, `P_{12}`, `u`, child(`w`))` returns the path produced by combining P_{11} and P_{12} through the edge $(u, \texttt{child}(w))$ of $(T + e)^3$. See the text for details.

is rooted at a vertex v_R and might be restricted not to contain a set of edges X and to contain a set of edges I. Here, v_R, X, and I can be quickly determined according to the classified subcase as described in the proof of the theorem. Then, Algorithm 2 is exploited to search for a paired 2-DPC of T^3, which will also be a solution for G^3.

For the type-B operation (Lines 18 to 24), a depth-first search starts at a carefully selected vertex v_R, finding a spanning tree T as well as a vertex u from which a back edge to the root vertex v_R exists. Then, Algorithm 1 (`Hamiltonian_Path_in_TREE_CUBE`) is performed for a terminal vertex w in $A \cup B$ to build a Hamiltonian path in $T^3_{(w)}$ from a vertex w to $\texttt{child}(w)$, a child of w. In addition, Algorithm 2 is used to find a paired 2-DPC in the cube of $T \setminus T_{(w)}$ joining two terminal sets that are slightly changed from A and B by replacing w with u. Finally, the Hamiltonian path and the paired 2-DPC are merged via the `CombinePaths` function to produce a paired 2-DPC of $(T+e)^3$ with $e = (v_R, u)$, joining the input terminal sets A and B, which will also be a solution for G^3. Recall from the proof of Theorem 4 that $v_R = \alpha_1$ and $w = \beta_1$ in Case 3-2-b, $v_R = \alpha_2$ and $w = \alpha_1$ in Case 3-3-b-ii, and v_R is a non-terminal vertex such that $A \cup B \subseteq N_G[v]$ and $w = \alpha_1$ in Case 4-3-b-ii. With these setups, the call to the function $\texttt{CombinePaths(}P_{11}\texttt{, }P_{12}\texttt{, }u\texttt{, child(}w\texttt{))}$ returns the path obtained by concatenating P_{11} and P_{12} through the edge $(u, \texttt{child(}w\texttt{)})$ of $(T+e)^3$.

Theorem 5. *Given a connected graph G with $n\,(\geq 4)$ vertices and m edges and two terminal sets $A = \{\alpha_1, \alpha_2\}$ and $B = \{\beta_1, \beta_2\}$, we can determine the existence of a paired 2-DPC of G^3 joining A and B and find one if one exists in $O(n+m)$ time.*

Proof. For efficient implementation of Algorithm 3, we adopt an adjacency list structure to represent the graph G. Then, the computation to identify bridges and cut vertices of G (Lines 2 and 3) can be done in $O(n+m)$ time using the well-known algorithm for computing biconnected components [14]. Also, the existence of a desired 2-DPC can also be determined (Line 4) in $O(n+m)$ time by examining the connectivity of the four terminal vertices and the properties of the edges incident to them. While the question in Line 6 can be answered in constant time, Line 10 takes $O(n)$ time to identify the structure of H, in which the decision of a proper operation to perform and the relevant information is determined in constant time based on the precomputed information. The remaining expensive operations are the depth-first search which can be done in $O(n+m)$ time, and the computations for finding a Hamiltonian path and a paired 2-DPC in the cube of a tree which can be done in $O(n)$ time by Corollary 1 and Theorem 3. Therefore, a paired 2-DPC of G^3 joining A and B can be built in linear time if it exists. $\square$

5. Concluding remarks

In this paper, we have presented a linear-time algorithm to find a paired 2-DPC in the cube of a connected graph joining two arbitrary terminal sets. In finding a Hamiltonian cycle or path in the cube of a connected graph in linear time, it was sufficient to examine only the edges of an arbitrary spanning tree of the graph. Interestingly, we have shown that almost the same number of edges were enough to find a paired 2-DPC in linear time. For the latter problem, however, the spanning tree and possibly an additional edge had to be selected carefully so that the skeleton graph is *feasible* for the input terminal sets. Our paired 2-DPC algorithm easily leads not only to a linear-time algorithm for finding an unpaired 2-DPC in the cube of a connected graph, but also to a linear-time algorithm for fining a Hamiltonian path in the cube of a connected graph that should pass through a prescribed edge (x, y), whether the direction between x and y in the path is specified or not. Unlike the cube of a connected graph that allows a solid construction, finding a paired 2-disjoint path cover in the square of a 2-connected graph still remains a challenging problem.

ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea (NRF) funded by the Korea government (MSIP and MOE) (Grant Nos. 2015R1A2A2A01006590 and 2015R1D1A1A09056849).

References

[1] J.A. Bondy and U.S.R. Murty, *Graph Theory*, 2nd printing, Springer, 2008.

[2] R. Caha and V. Koubek, "Hamiltonian cycles and paths with a prescribed set of edges in hypercubes and dense sets," *Journal of Graph Theory* **51(2)**, pp. 137–169, 2006.

[3] M.-S. Chang, M.-T. Ko, and H.-I Lu, "Linear-time algorithms for tree root problems," in *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT'06)*, pp. 411–422, 2006.

[4] G. Chartrand, A.M. Hobbs, H.A. Jung, S.F. Kapoor, and C.St.J.A. Nash-Williams, "The square of a block is Hamiltonian connected," *Journal of Combinatorial Theory, Series B* **16**, pp. 290–292, 1974.

[5] G. Chartrand and S.F. Kapoor, "The cube of every connected graph is 1-Hamiltonian," *Journal of Research of the National Bureau of Standards* **73B**, pp. 47–48, 1969.

[6] X.-B. Chen, "Many-to-many disjoint paths in faulty hypercubes," *Information Sciences* **179(18)**, pp. 3110–3115, 2009.

[7] G.L. Chia, S.-H. Ong, and L.Y. Tan, "On graphs whose square have strong Hamiltonian properties," *Discrete Mathematics* **309(13)**, pp. 4608–4613, 2009.

[8] T. Dvořák and P. Gregor, "Hamiltonian paths with prescribed edges in hypercubes," *Discrete Mathematics* **307**, pp. 1982–1998, 2007.

[9] T. Dvořák and P. Gregor, "Partitions of faulty hypercubes into paths with prescribed endvertices," *SIAM Journal on Discrete Mathematics* **22(4)**, pp. 1448–1461, 2008.

[10] F. Escalante, L. Montejano, and T. Rojano, "Characterization of n-path graphs and of graphs having nth root," *Journal of Combinatorial Theory, Series B* **16**, pp. 282–289, 1974.

[11] H. Fleischner, "The square of every two-connected graph is Hamiltonian," *Journal of Combinatorial Theory, Series B* **16**, pp. 29–34, 1974.

[12] A. Georgakopoulos, "A short proof of Fleischner's theorem," *Discrete Mathematics* **309**, pp. 6632–6634, 2009.

[13] P. Gregor and T. Dvořák, "Path partitions of hypercubes," *Information Processing Letters* **108(6)**, pp. 402–406, 2008.

[14] J.E. Hopcroft and R.E. Tarjan, "Efficient algorithms for graph manipulation (Algorithm 447)," *Communications of the ACM* **16(6)**, pp. 372–378, June 1973.

[15] S. Jo, J.-H. Park, and K.Y. Chwa, "Paired many-to-many disjoint path covers in faulty hypercubes," *Theoretical Computer Science* **513**, pp. 1–24, Nov. 2013.

[16] J.J. Karaganis, "On the cube of a graph," *Canadian Mathematical Bulletin* **11**, pp. 295–296, 1968.

[17] P.E. Kearney and D.G. Corneil, "Tree powers," *Journal of Algorithms* **29**, pp. 111–131, 1998.

[18] S.-Y. Kim, J.-H. Lee, and J.-H. Park, "Disjoint path covers in recursive circulants $G(2^m, 4)$ with faulty elements," *Theoretical Computer Science* **412(35)**, pp. 4636–4649, 2011.

[19] S.-Y. Kim and J.-H. Park, "Paired many-to-many disjoint path covers in recursive circulants $G(2^m, 4)$," *IEEE Transactions on Computers* **62(12)**, pp. 2468–2475, Dec. 2013.

[20] K.M. Koh and K.L. Teo, "The 2-Hamiltonian cubes of graphs," *Journal of Graph Theory* **13(6)**, pp. 737–747, 1989.

[21] L.C. Lau, "Bipartite roots of graphs," in *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pp. 952–961, 2004.

[22] L.C. Lau and D.G. Corneil, "Recognizing powers of proper interval, split, and chordal graph," *SIAM Journal on Discrete Mathematics* **18(1)**, pp. 83–102, 2004.

[23] L. Lesniak, "Graphs with 1-Hamiltonian-connected cubes," *Journal of Combinatorial Theory, Series B* **14(2)**, pp. 148–152, 1973.

[24] Y.-L. Lin and S. Skiena, "Algorithms for square roots of graphs," *SIAM Journal on Discrete Mathematics* **8(1)**, pp. 99–118, 1995.

[25] M. Milanič, A. Oversberg, and O. Schaudt, "A characterization of line graphs that are squares of graphs," *Discrete Applied Mathematics* **173**, pp. 83–91, 2014.

[26] M. Milanič and O. Schaudt, "Computing square roots of trivially perfect and threshold graphs," *Discrete Applied Mathematics* **161(10–11)**, pp. 1538–1545, 2013.

[27] R. Motwani and M. Sudan, "Computing roots of graphs is hard," *Discrete Applied Mathematics* **54(1)**, pp. 81–88, 1994.

[28] A. Mukhopadhyay, "The square root of a graph," *Journal of Combinatorial Theory* **2(3)**, pp. 290–295, 1967.

[29] J. Müttel and D. Rautenbach, "A short proof of the versatile version of Fleischner's theorem," *Discrete Mathematics* **313(19)**, pp. 1929–1933, 2013.

[30] M. Paoli, "Hamiltonian properties of the cube of a 2-edge connected graph," *Journal of Graph Theory* **12(1)**, pp. 85–94, 1988.

[31] J.-H. Park and I. Ihm, "Single-source three-disjoint path covers in cubes of connected graphs," *Information Processing Letters* **113(14–16)**, pp. 527–532, 2013.

[32] J.-H. Park and I. Ihm, "Disjoint path covers in cubes of connected graphs," *Discrete Mathematics* **325**, pp. 65–73, 2014.

[33] J.-H. Park and I. Ihm, "Many-to-many two-disjoint path covers in cylindrical and toroidal grids," *Discrete Applied Mathematics* **185**, pp. 168–191, 2015.

[34] J.-H. Park, H.-C. Kim, and H.-S. Lim, "Many-to-many disjoint path covers in hypercube-like interconnection networks with faulty elements," *IEEE Transactions on Parallel and Distributed Systems* **17(3)**, pp. 227–240, 2006.

[35] J.-H. Park, H.-C. Kim, and H.-S. Lim, "Many-to-many disjoint path covers in the presence of faulty elements," *IEEE Transactions on Computers* **58(4)**, pp. 528–540, 2009.

[36] G. Schaar, "On 3-Hamiltonian cubes of connected graphs," *Colloquia Mathematica Societatis János Bolyai* **18**, pp. 959–971, 1978.

[37] M. Sekanina, "On an ordering of the set of vertices of a connected graph," *Publication of the Faculty of Sciences of the University of Brno* **412**, pp. 137–142, 1960.