A linear-time algorithm for finding a one-to-many 3-disjoint path cover in the cube of a connected graph

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

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

Abstract

Given two disjoint vertex sets $S = \{s\}$ and $T = \{t_1, t_2, t_3\}$ of a connected graph, a one-to-many 3-disjoint path cover joining S and T is a vertex-disjoint path cover $\{P_1, P_2, P_3\}$ such that each path P_i joins s and t_i. In this paper, we present an efficient algorithm that builds, if exists, a one-to-many 3-disjoint path cover in the cube of a connected graph G joining the two terminal sets. Interestingly enough, we show that a carefully selected spanning tree or spanning unicyclic subgraph of G is all we need to find one in time linear to the size of G.

Keywords: Disjoint path cover, cube of graph, graph algorithms.

1. Introduction

In this paper, we only consider an undirected graph G with vertex set $V(G)$ and edge set $E(G)$. For two vertices u and v of G, a *path* from u to v is a sequence $\langle w_1, \ldots, w_l \rangle$ of distinct vertices of G such that $w_1 = u$, $w_l = v$, and $(w_i, w_{i+1}) \in E(G)$ for $i \in \{1, \ldots, l-1\}$. If $l \geq 3$ and $(w_l, w_1) \in E(G)$, the sequence forms a *cycle*. A path that visits every vertex of G exactly once is a *Hamiltonian path* of G; a cycle that visits every vertex of G exactly once is a *Hamiltonian cycle* of G. A *path cover* of G is a set of paths in G such that every vertex of G belongs to at least one path. A *disjoint path cover* of G is a set of internally vertex-disjoint paths that altogether cover every vertex of G.

For two disjoint vertex sets $S = \{s\}$ and $T = \{t_1, \ldots, t_k\}$ of G, the *one-to-many k-disjoint path cover* (k-*DPC* for short) is a disjoint path cover composed of k paths, each of which respectively joins the pair of s and t_i, $i \in \{1, \ldots, k\}$. When $S = \{s\}$ and $T = \{t\}$, a disjoint path cover made of k paths, each joining s and t, is called a *one-to-one k-DPC*. Another interesting disjoint path cover is the *many-to-many k-DPC*, whose k disjoint paths collectively join disjoint vertex sets $S = \{s_1, \ldots, s_k\}$ and $T = \{t_1, \ldots, t_k\}$; if each source s_i is designated to a specific sink t_i, the disjoint path cover is called *paired*, or *unpaired* otherwise. As intuitively clear, we will call the vertices in S and T *sources* and *sinks*, respectively, which together form a set of *terminals*.

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

Various forms of disjoint path cover problems are found in many areas like code optimization, software testing, and database design [1, 11]. Interestingly, the existence of a disjoint path cover in a graph is closely related to the Hamiltonian properties as well as the concept of vertex connectivity, characterized with respect to the minimum number of disjoint paths. In a graph with at least three vertices, for instance, a t_1–t_2 Hamiltonian path forms a one-to-many 2-DPC joining $\{s\}$ and $\{t_1, t_2\}$ for any vertex $s \neq t_1, t_2$; a Hamiltonian cycle forms a one-to-one 2-DPC joining $\{s\}$ and $\{t\}$ for any pair of distinct vertices s and t. Deciding the existence of a k-DPC in a general graph, however, is NP-complete for any fixed $k \geq 1$ [14, 15]. Thus, researches have studied the existence for some specific classes of graphs such as hypercubes [4], hypercube-like graphs [14, 15], and tori [3]. For the cube of a connected graph, the exact condition for a paired 2-DPC joining two terminal sets to exist was presented by the authors in [13], and, based upon the result, a linear-time algorithm that finds such a paired 2-DPC was developed in [6]. Also, a characterization was established in [12] for the existence of a one-to-many 3-DPC joining terminal sets in the cube of a connected graph.

In this paper, we present a linear-time algorithm that builds, if exists, a one-to-many 3-DPC joining two terminal sets in the cube of a connected graph. For this, we carefully and efficiently select a spanning tree or a spanning unicyclic subgraph of the graph, for which another, simpler one-to-many 3-DPC algorithm of ours that runs in time linear to the size of the graph is explored.

2. Preliminaries

For a positive integer d, the *d-th power* G^d of a graph G is the graph, defined over the same vertex set as G, whose two vertices are adjacent to each other if and only if there exists a path of length at most d in G joining them. If a graph H is the d-th power of G, i.e. $H = G^d$, then G is called a d-th *root* of H. When $d = 3$, we call H and G the *cube* of G and a *cube root* of H, respectively. Determining whether a graph has a d-th root is not easy in general [10]; for some classes of graphs, polynomial-time algorithms to solve the root finding problem have been investigated [2, 8]. Note that if a cube root G of a given graph H could be found in polynomial time such that $H = G^3$, then we can solve the one-to-many 3-DPC problem on the graph H also in polynomial time by combining our algorithm, presented in this paper, that finds a one-to-many 3-DPC of G^3 in linear time.

To understand the exact conditions for the cube of a connected graph to have a one-to-many 3-DPC joining given terminal sets, we first give some definitions: A *cut vertex* and *bridge* of a graph are respectively a vertex and an edge whose removal increases the number of connected components in the graph. A *nontrivial* bridge is a bridge both of whose endvertices have degrees greater than one (refer to Fig. 1). Note that a bridge is *trivial* if any of its endvertices is of degree one implying deleting the bridge leaves a one-vertex component in the graph. A vertex of G is said to be a *pure bridge vertex* if each of its incident edges is a nontrivial bridge. Also, a set of three mutually adjacent vertices, each having a degree of at least three, is called a *pure bridge triangle* if every edge that is incident with exactly one of the triangular vertices is a nontrivial bridge. In addition, $N_G[u]$, or $N[u]$ if the graph G is clear in the context, denotes the closed neighborhood of a vertex u, i.e. $N_G[u] = \{u\} \cup \{v \in V(G) : (u, v) \in E(G)\}$. Furthermore, we define $N_G[W] = \bigcup_{w \in W} N_G[w]$ for $W \subseteq V(G)$.

Theorem 1 ([12])**.** *Given disjoint terminal sets $S = \{s\}$ and $T = \{t_1, t_2, t_3\}$ of a connected graph G, the cube G^3 has a one-to-many 3-DPC joining S and T if and only if*

C1: *there exists no pure bridge vertex u in G such that $T \subseteq N_G[u]$ and $s \notin N_G[u]$, and*

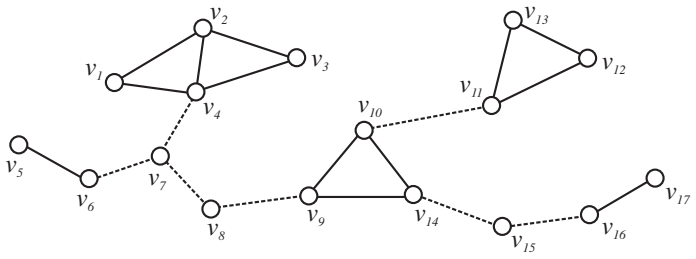


Fig. 1: Some terminologies. The seven *nontrivial bridges* are marked in dotted lines. Also, there are three *pure bridge vertices* v_7, v_8, and v_{15}, and one *pure bridge triangle* $\{v_9, v_{10}, v_{14}\}$.

C2: *T does not form a pure bridge triangle in G such that $s \notin N_G[T]$.*

Note that the validity of the two conditions in the above theorem can be determined in linear time basically by identifying all nontrivial bridges of the graph with a help of the biconnected component algorithm [5]. Now, we describe two more theorems on the Hamiltonian properties of the cube of a connected graph, which will play key roles in developing our algorithms.

Theorem 2 ([7, 16])**.** *The cube of a connected graph is Hamiltonian-connected, i.e. every pair of distinct vertices are joined by a Hamiltonian path.*

Theorem 3 ([6, 9])**.** *Given a pair of distinct vertices in a tree of order n, a Hamiltonian path joining them in the cube of the tree can be found in $O(n)$ time.*

Finally, throughout this paper, we denote by T_u a tree rooted at vertex u. Also, $P \circ Q$ denotes the concatenation of two paths P and Q, i.e. $P \circ Q = \langle u_1, \ldots, u_p, v_1, \ldots, v_q \rangle$ for $P = \langle u_1, \ldots, u_p \rangle$ and $Q = \langle v_1, \ldots, v_q \rangle$.

3. Finding a one-to-many 3-DPC in the cube of a tree

In this section, we describe a linear-time algorithm for finding a one-to-many 3-DPC joining disjoint terminal sets $S = \{s\}$ and $T = \{t_1, t_2, t_3\}$ in the cube of a tree T_s rooted at the source s. Throughout this section, when we refer to Theorem 1, we restrict our attention to the *C1* condition only because the *C2* condition is always valid for trees. Furthermore, since the validity of the conditions can be decided in linear time with respect to the size of the graph, we only handle the feasible case in which the *C1* condition is satisfied, and thus a desired one-to-many 3-DPC exists. Note that every edge of a tree is a bridge; an edge incident with a leaf or incident with the root having a single child is trivial whereas all other edges are nontrivial.

Before presenting the main algorithm of this section, we first solve a subproblem of finding a special type of Hamiltonian path defined as follows: *Given a tree T_u rooted at vertex u and a prescribed child z of u, find a Hamiltonian path of $T_u^3 - u$ that runs from some vertex v to z such that v is a child of u if u has a child that is a leaf, or a grandchild of u otherwise.* Such v–z Hamiltonian path always exists, and Algorithm 1 describes a way to find a desired path by combining Hamiltonian paths in the cubes of the subtrees of T_u rooted at the children of u, where the vertex v is determined differently whether u has a child that is a leaf or not. Note that this algorithm uses Theorem 3, assuring that there is a function `HP_in_TCUBE(`T`,`u`,`v`)` that finds a

Algorithm 1: Finding a Hamiltonian path in $T_u^3 - u$

```
1  Function HP_in_TCUBE-ROOT(T_u, z)
2      flag ← false ;              // a leaf child of u has not yet been found
3      if z is a leaf then P ← ⟨z⟩; flag ← true;
4      else P ← HP_in_TCUBE(T_z, w, z) for a child w of z;
5      foreach child c of u other than z do
6          if c is a leaf then P' ← ⟨c⟩; flag ← true;
7          else
8              w ← a child of c;
9              if flag = true then P' ← HP_in_TCUBE(T_c, c, w);
10             else P' ← HP_in_TCUBE(T_c, w, c);
11         P ← P' ∘ P ;              // place the new path P' in front of P
12     end
13     return P;
14 end
```

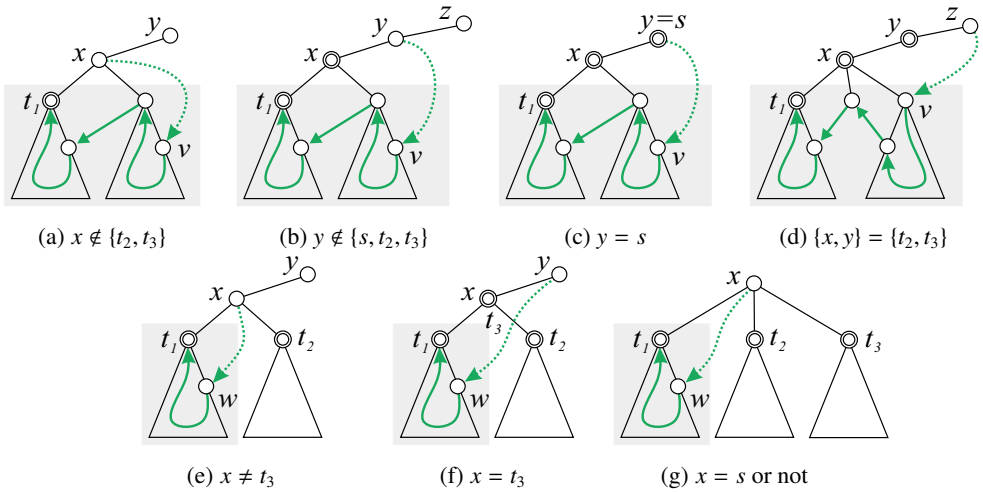
Hamiltonian path joining two distinct vertices u and v in the cube of a tree T in time linear to the size of T.

Lemma 1. *Let T_u be a tree rooted at vertex u. Then for a child z of u,* `HP_in_TCUBE-ROOT`(T_u, z) *in Algorithm 1 finds a Hamiltonian path of $T_u^3 - u$ that runs from a vertex v to z where v is a child of u if u has a child that is a leaf, or a grandchild of u otherwise. Furthermore, it runs in time linear to the size of T_u.*

Proof. The proof is obvious from Theorem 3. ■

The main algorithm of this section that finds a one-to-many 3-DPC in T_s^3 for a rooted tree T_s and terminal sets $S = \{s\}$ and $T = \{t_1, t_2, t_3\}$ is inherently recursive in that the given one-to-many 3-DPC problem is dissected into two subproblems: a Hamiltonian path problem and (recursively) a one-to-many 3-DPC problem on a smaller graph, where the second subproblem is sometimes replaced by a simple Hamiltonian path problem. The key to our algorithm is how to divide the input graph T_s^3 into a pair of cube graphs on which the two subproblems are solved respectively. For this, define `numSinks[`u`]` for a vertex u to be the number of sinks that are proper descendants of u, where it is trivial that `numSinks[`s`] = 3` and `numSinks[`u`] = 0` if u is a leaf. Now, from all parents of the sinks, choose a *pivot* vertex x which has the smallest `numSinks` value, i.e. `numSinks[`x`]` $\leq$ `numSinks[`x'`]` for all other parents x' of sinks. Then, depending on the value of `numSinks[`x`]`, the pair of cubes we seek are either $T_x^3 - x$ and the cube of $T'_s := T_s - (T_x - x)$, or $T_{t_i}^3$ and the cube of $T'_s := T_s - T_{t_i}$ for some sink child t_i of x, as will be detailed below.

Our algorithm is as follows (refer to Fig. 2). First, consider the case of `numSinks[`x`] = 1`, where the sink t_1 is assumed to be a child of x. Let P be a v–t_1 Hamiltonian path in $T_x^3 - x$ found by the function `HP_in_TCUBE-ROOT`(T_x, t_1), where, as noted earlier, v is either a child or a grandchild of x depending on whether x has a child that is a leaf or not. Also, let T'_s denote the subtree of T_s with all proper descendants of x being deleted, i.e. $T'_s = T_s - (T_x - x)$. If x is not a sink (Fig. 2(a)), we find a one-to-many 3-DPC joining $\{s\}$ and $\{x, t_2, t_3\}$ in the cube of T'_s which exists because x, one of the three sinks, is a leaf of T'_s, thus satisfying the *C1* condition of



(a) $x \notin \{t_2, t_3\}$ (b) $y \notin \{s, t_2, t_3\}$ (c) $y = s$ (d) $\{x, y\} = \{t_2, t_3\}$

(e) $x \neq t_3$ (f) $x = t_3$ (g) $x = s$ or not

Fig. 2: Hybrid subcases in the description of Algorithm 2: (a) to (d) for `numSinks[`x`]` = 1, (e) and (f) for `numSinks[`x`]` = 2, and (g) for `numSinks[`x`]` = 3.

Theorem 1. (In order to break the *C1* condition, every sink must, at least, be either a pure bridge vertex or adjacent to a pure bridge vertex, which cannot be true for a sink that is a leaf.) Then, a desired one-to-many 3-DPC in T_s^3 can be built by combining the s–x path and P.

If x happens to be a sink, consider the parent of x, named y, which exists since the pivot x is not the root of T_s. If y is not a terminal (Fig. 2(b)), a desired 3-DPC can be built analogously from P and a one-to-many 3-DPC in the cube of T'_s joining $\{s\}$ and $\{y, t_2, t_3\}$. If y is a source (Fig. 2(c)), then $\langle s \rangle \circ P$ forms an s–t_1 path, in which case it suffices to build s–t_2 and s–t_3 paths from a t_2–t_3 Hamiltonian path in the cube of T'_s. If y is also a sink, meaning that $\{x, y\} = \{t_2, t_3\}$, $t_1, y \in N[x]$, and y also has a parent, named z (Fig. 2(d)), then x must have a child which is a leaf; otherwise, x would be a pure bridge vertex breaking the *C1* condition of Theorem 1 in T_s. This implies that the start vertex of P, the v–t_1 Hamiltonian path in $T_x^3 - x$, may be reached from z in T_s^3. Thus, if z is not a source, a desired DPC can be built from P and a one-to-many 3-DPC joining $\{s\}$ and $\{z, t_2, t_3\}$ in the cube of T'_s; if z is a source, it can be constructed from P and a t_2–t_3 Hamiltonian path of the cube of T'_s.

Second, consider the other two cases of `numSinks[`x`]` $\in \{2, 3\}$, where at least two of the three sinks are children of the pivot x (recall the choice of the pivot x). Let them be t_1 and t_2, respectively. In particular, we assume t_1 to be the sink that has a child if there is such one (this choice of t_1 will guarantee the existence of a one-to-many 3-DPC that we will seek in a subtree later). Let P be a w–t_1 Hamiltonian path of $T_{t_1}^3$ produced by `HP_in_TCUBE()` if t_1 has a child w; $P = \langle t_1 \rangle$ otherwise. On the other hand, let T'_s be the subtree of T_s with the vertices of T_{t_1} being deleted, i.e. $T'_s = T_s - T_{t_1}$.

If `numSinks[`x`]` = 2, x cannot be the source s, implying it has a parent, named y. There are two subcases to consider. Assume x is not a sink (Fig. 2(e)). Then, for the sink set $\{x, t_2, t_3\}$, x may not be a pure bridge vertex of T'_s such that $x, t_2, t_3 \in N[x]$ and $s \notin N[x]$ because otherwise t_1 as well as t_2 would have a child (recall how t_1 has been chosen), indicating x is a pure bridge vertex of T_s such that $t_1, t_2, t_3 \in N[x]$ and $s \notin N[x]$, breaking the *C1* condition for T_s. Therefore, there exists a one-to-many 3-DPC joining $\{s\}$ and $\{x, t_2, t_3\}$ in the cube of T'_s, which builds a

Algorithm 2: Finding a one-to-many 3-DPC in the cube of a tree

Function `3-DPC_in_TCUBE`$(T_s, \{s\}, \{t_1, t_2, t_3\})$
  $x \leftarrow$ a pivot vertex;
  if `numSinks[`x`]` $= 1$ **then** `// Assume` t_1 `is a child of` x`.`
    $P \leftarrow$ `HP_in_TCUBE-ROOT(`T_x, t_1`)`;
    $T'_s \leftarrow T_s - (T_x - x)$;
    if *x is not a sink* **then** $u \leftarrow x$;
    else if *the parent of x is not a sink* **then** $u \leftarrow$ the parent of x;
    else $u \leftarrow$ the grandparent of x;
  else `// Assume` t_1 `and` t_2 `are children of` x`.`
    `/* Assume further` t_1 `has a child or` t_2 `has no child. */`
    if t_1 *is a leaf* **then** $P \leftarrow \langle t_1 \rangle$;
    else $P \leftarrow$ `HP_in_TCUBE(`T_{t_1}, w, t_1`)` for a child w of t_1;
    $T'_s \leftarrow T_s - T_{t_1}$;
    if `numSinks[`x`]` $= 2$ **then**
      if *x is not a sink* **then** $u \leftarrow x$;
      else $u \leftarrow$ the parent of x;
    else $u \leftarrow x$; `// numSinks[`x`]` $= 3$
  if $u \neq s$ **then**
    $\mathcal{P}' \leftarrow$ `3-DPC_in_TCUBE(`$T'_s, \{s\}, \{u, t_2, t_3\}$`)`;
    $\mathcal{P} \leftarrow (\mathcal{P}' - P') \cup \{P' \circ P\}$, where P' is the s–u path in $\mathcal{P}'$;
  else
    $P' \leftarrow$ `HP_in_TCUBE(`T'_s, t_2, t_3`)`;
    Transform P' into a 2-DPC $\mathcal{P}'$ joining $\{s\}$ and $\{t_2, t_3\}$;
    $\mathcal{P} \leftarrow \mathcal{P}' \cup \{\langle s \rangle \circ P\}$;
  return $\mathcal{P}$;
end

desired 3-DPC together with the path P. If x is a sink, i.e. $x = t_3$ as illustrated in Fig. 2(f), a desired 3-DPC can be built using a one-to-many 3-DPC of the cube of T'_s joining $\{s\}$ and $\{y, t_2, t_3\}$ if $y \neq s$ (which is guaranteed to exist for the same reason as before), or using a t_2–t_3 Hamiltonian path of the cube of T'_s otherwise. If `numSinks[`x`]` $= 3$, all the three sinks t_1, t_2, and t_3 become children of x (Fig. 2(g)). Then, similarly as before, a desired 3-DPC can be built using a one-to-many 3-DPC of the cube of T'_s joining $\{s\}$ and $\{x, t_2, t_3\}$ if $x \neq s$ (which also exists thanks to the choice of t_1), or using a t_2–t_3 Hamiltonian path of the cube of T'_s otherwise. The resulting algorithm is now summarized in Algorithm 2.

Theorem 4. *Given a tree of order $n \geq 4$ and disjoint terminal sets $S = \{s\}$ and $T = \{t_1, t_2, t_3\}$ that satisfy the two conditions of Theorem 1, a one-to-many 3-DPC in the cube of the tree can be found in $O(n)$ time.*

Proof. Basically, our recursive algorithm finds the answer by computing a Hamiltonian path and a one-to-many 3-DPC in the cube of a subtree T'_s of T_s (or another Hamiltonian path). As verified above, the one-to-many 3-DPC always exists in the cube of T'_s because the given terminal sets satisfy the *C1* condition of Theorem 1 for T'_s. Also, the Hamiltonian path(s) can be found

correctly in linear time by Theorem 3 and Lemma 1. Therefore, the correctness of our algorithm can be shown by induction on the number of vertices of T_s, and is omitted here. (Note that the base case for a tree T_s with four vertices is trivial, and the inductive step for the tree T_s having more than four vertices relies on the inductive hypothesis that the theorem is valid for its subtree T'_s having a fewer number of vertices.) Time complexity-wise, the cost of maintaining the `numSinks` values, which is essential for choosing the pivot vertex, must be analyzed: For the initial tree T_s with n vertices, the entire array is initialized once in $O(n)$ time before the algorithm starts. Then, each time the algorithm is called recursively on a smaller tree T'_s, it is enough to update the `numSinks` values for a constant number of vertices of T'_s. (For example, in the case illustrated in Fig. 2(d) with $T'_s = T_s - (T_x - x)$, it suffices to decrease by one the current value for each of x, y, and the new terminal z.) This means that each pivot vertex can be selected in constant time. Also, it is trivial that other minor operations is done in constant time. Therefore, the run time $T(n)$ of Algorithm 2 is $\max_{1 \le n' < n}\{O(n') + T(n - n'), O(n') + O(n - n')\} + c$ for some constant c, leading to $T(n) \le 2cn - c$. ■

4. Finding a one-to-many 3-DPC in the cube of a connected graph

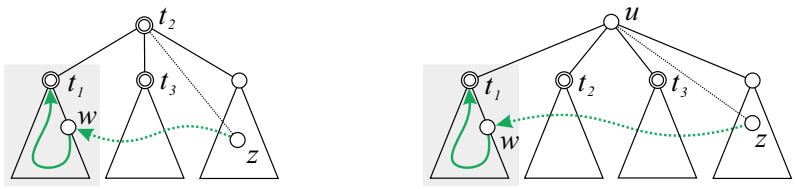
Now, we are ready to describe our linear-time algorithm for finding a one-to-many 3-DPC in the cube of a connected graph G joining disjoint terminal sets $S = \{s\}$ and $T = \{t_1, t_2, t_3\}$. Similarly as before, we assume that the two conditions *C1* and *C2* of Theorem 1 have been validated in time linear to the size of G. The main idea of our algorithm is to extract a simple subgraph from G, either a spanning tree or a spanning unicyclic subgraph, from whose cube a desired one-to-many 3-DPC can efficiently be built for G^3. In the following theorem, we show that it is always possible to find such a spanning subgraph.

Theorem 5. *Given a connected graph G having at least four vertices and disjoint terminal sets $S = \{s\}$ and $T = \{t_1, t_2, t_3\}$ that satisfy the two conditions of Theorem 1, there exists a rooted spanning tree T_u of G such that T_u^3 or $(T_u + e)^3$ for some edge $e \in E(G) \setminus E(T_u)$ admits a one-to-many 3-DPC joining S and T.*

Proof. Our proof proceeds by induction on the number of edges of G. Since the base case can be proven trivially, we focus on the inductive step. If some sink t_i is not a cut vertex of G, consider a depth-first-search tree (a DFS tree in short) T_{t_i} rooted at t_i. Then it is clear that the root t_i has only one child in T_{t_i}, and thus is connected to the rest of T_{t_i} via a trivial bridge, guaranteeing the validity of the two conditions of Theorem 1 for T_{t_i}. Thus, $T_{t_i}^3$ (and thus G^3) will admit a one-to-many 3-DPC joining S and T. Thus hereafter we assume that every sink is a cut vertex of G. Now, let H be the subgraph of G induced by the three sinks. Then there are four cases.

Case 1: $|E(H)| = 1$. Let T_u be a DFS tree of G containing the edge of H, which is easily built by placing the edge in the front of the respective lists in the adjacency list representation of G. Then, the sink that is not incident to the only edge of H may not be incident to the other sinks in T_u, implying both *C1* and *C2* conditions are satisfied for T_u. (Recall that in order to break the conditions, the three sinks must, at least, either belong to the closed neighborhood of a vertex or form a triangle in the tree.) So T_u^3 admits a desired one-to-many 3-DPC.

Case 2: $|E(H)| = 2$. Assume w.l.o.g. that $E(H) = \{(t_1, t_2), (t_2, t_3)\}$. Suppose an edge of H, say (t_1, t_2), is not a bridge of G. The terminal sets still satisfy the $C1$ condition in the connected graph $G - (t_1, t_2)$ as a pure bridge vertex cannot include the three sinks in its closed neighborhood. They also satisfy the $C2$ condition in $G - (t_1, t_2)$ as the sinks even may not form a triangle. Thus for



(a) The last subcase of Case 2 (b) The last subcase of Case 4

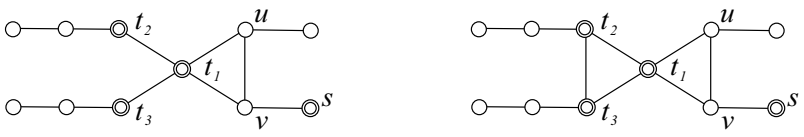
Fig. 3: Two hybrid subcases in the proof of Theorem 5.

$G - (t_1, t_2)$, which has one fewer edges than G, the theorem holds by the inductive hypothesis, which is also true for G. (Recall that the proof is inductive on the number of edges of the graph, and the current case is reduced to Case 1 of $|E(H)| = 1$ in the smaller graph.)

So, we assume for this case that both (t_1, t_2) and (t_2, t_3) are bridges, which are also nontrivial because all the sinks are cut vertices as assumed. If $s \in N[t_2]$, the validity of the two conditions clearly continues to be true in any spanning tree containing (s, t_2) as well as the two edges of H. Now, assume further $s \notin N[t_2]$. Note that t_2 cannot be a pure bridge vertex of G by the hypothesis of the theorem, implying, in G, either (i) some vertex adjacent to t_2 has degree 1 or (ii) some edge incident with t_2 is not a bridge. For the subcase (i), it suffices to build a spanning tree that contains the edges of H. For the subcase (ii), we build a spanning unicyclic subgraph whose cube admits a wanted disjoint path cover. Performing a depth-first search from t_2 returns a DFS tree T_{t_2}, in which there is at least one back edge (z, t_2) from a vertex z to t_2. (Note that a back edge is an edge from a vertex to one of its ancestors in the tree produced by a DFS.) Then, since (t_2, t_1) and (t_2, t_3) are nontrivial bridges, z cannot belong to the rooted subtrees T_{t_1} and T_{t_3} of T_{t_2}. Assuming safely s does not belong T_{t_1} (we can always name the sink whose corresponding subtree does not contain s as t_1), let P be a w–t_1 Hamiltonian path of $T^3_{t_1}$ for some child w of t_1 (see Fig. 3(a)). Then, based on the fact that (z, w) is an edge of the cube of the spanning unicyclic subgraph $G' := T_{t_2} + (z, t_2)$, a one-to-many 3-DPC in the cube of G' can be built from P and a one-to-many 3-DPC of the cube of $T_{t_2} - T_{t_1}$ joining $\{s\}$ and $\{z, t_2, t_3\}$ if $z \neq s$; from P and a t_2–t_3 Hamiltonian path of the cube of $T_{t_2} - T_{t_1}$ otherwise.

Case 3: $|E(H)| = 3$. By the hypothesis of the theorem, $\{t_1, t_2, t_3\}$ does not form a pure bridge triangle in G such that $s \notin N[\{t_1, t_2, t_3\}]$. Thus, there must be a sink, say t_1, such that (i) no edge other than (t_1, t_2) and (t_1, t_3) is incident with t_1, (ii) there is an edge incident with t_1 other than (t_1, t_2) and (t_1, t_3) that is not a nontrivial bridge, or (iii) $s \in N(t_1)$. For the graphs $G - (t_1, t_2)$ for the subcase (i) and $G - (t_2, t_3)$ for the subcases (ii) & (iii), the terminal sets still satisfy the *C1* condition (trivially *C2* too), which recursively sends us to Case 2 of a graph having fewer edges than G.

Case 4: $|E(H)| = 0$. Since, in this case, a subgraph of G may break the *C1* condition only, we focus on the necessary condition that a vertex u exists such that $\{t_1, t_2, t_3\} \subseteq N[u]$. If no such vertex exists, any spanning tree of G will be good. Hence, for the rest of the proof we assume that there exists such a vertex u. There are two cases. First, if some edge joining u and a sink, say (u, t_1), is not a bridge, we find a DFS tree T_{t_1} in $G - (u, t_1)$ starting from t_1. Then, it is easy to see that any other vertex v cannot satisfy the necessary condition in T_{t_1}, indicating T_{t_1} is the spanning subgraph we are looking for. Second, if the edges from u to the sinks, i.e. (u, t_1), (u, t_2), and (u, t_3) are all bridges, in particular, nontrivial because the sinks are all cut vertices as assumed in the beginning of the proof, we build a DFS tree T_u starting from u, where a back



(a) (b)

Fig. 4: Examples of hard-to-handle cases. Removing any edge of the triangle $\langle t_1, u, v\rangle$ in each graph results in violating the conditions *C1* and *C2*, respectively.

edge to the root u might has been found during the depth-first search. If no such back edge has been found, implying every edge incident with u is a bridge, then, to break the *C1* condition for G, either a trivial bridge must be incident with u or $s \in N[u]$. In either case, both *C1* and *C2* conditions are also valid for T_u. Finally, suppose a back edge to u, say (z, u) has been found (see Fig. 3(b)). Again, assuming safely s does not belong to the subtree T_{t_1} of T_u, rooted at t_1 (we can always name the sink whose corresponding subtree does not contain s as t_1), we find a w–t_1 Hamiltonian path P of $T^3_{t_1}$ for some child w of t_1. Then a desired disjoint path cover in the cube of the spanning unicyclic subgraph $T_u + (z, u)$ can be built from P and a one-to-many 3-DPC of the cube of $T_u - T_{t_1}$ joining $\{s\}$ and $\{z, t_2, t_3\}$ if $z \neq s$; from P and a t_2–t_3 Hamiltonian path of the cube of $T_u - T_{t_1}$ otherwise. This completes the entire proof. ■

Remark. Sometimes, the edges of a spanning tree T_u of G, however carefully selected, do not provide sufficient connectivity to guarantee the existence of a one-to-many 3-DPC in T^3_u (see Fig. 4 for some examples). Even in that case, however, an extra edge is always enough to resolve the problem.

The proof of Theorem 5 is constructive in nature. Thus, it provides an effective algorithm `3-DPC_in_GCUBE` for our problem, as summarized in Algorithm 3, where the function `DFS`(G, x) finds a DFS tree T_x rooted at x and the endvertex z of a back edge (z, x), if any found.

Theorem 6. *Given disjoint terminal sets $S = \{s\}$ and $T = \{t_1, t_2, t_3\}$ in a connected graph G with n vertices and m edges, we can both determine the existence of a one-to-many 3-DPC of G^3 joining S and T and find one if any exists in $O(n + m)$ time.*

Proof. If we represent the graph G using an adjacency list structure, all bridges and cut vertices of G can be identified in $O(n+m)$ time using the bicomponent algorithm, from which the validity of the *C1* and *C2* conditions may be determined also in $O(n + m)$ time. On the other hand, the correctness of Algorithm 3, `3-DPC_in_GCUBE()`, is direct from Theorems 3, 4, and 5. The time complexity is dominated by a recursive function call, the function calls to `HP_in_TCUBE()` and `3-DPC_in_TCUBE()`, both of which run in $O(n)$ time, and other basic operations, all of which can be done in linear time: (i) `DFS()` that returns a DFS tree, a back edge to the root, if any, and an indicator on $s \notin V(T_{t_1})$, (ii) finding a spanning tree containing a constant number of prescribed edges, (iii) transforming a tree into T_s, (iv) deleting an edge from G, (v) selecting an edge (t_i, t_j) in Step 15, (vi) choosing a vertex u in Step 18, if any, (vii) answering the question in Step 29, and finally (viii) transforming a path P' into a one-to-many 2-DPC $\mathcal{P}'$ in Step 41. Therefore, Algorithm 3 runs on an arbitrary connected graph and terminal sets in time

$$T(n, m, |E(H)|) \leq \begin{cases} T(n, m-1, |E(H)|-1) + O(n+m) & \text{if } |E(H)| \geq 2, \\ O(n+m) & \text{otherwise,} \end{cases}$$

which is easily shown to be $O(n + m)$. ■

Algorithm 3: Finding a one-to-many 3-DPC in the cube of a graph

Function `3-DPC_in_GCUBE`$(G, \{s\}, \{t_1, t_2, t_3\})$
  if *a sink t_i is not a cut vertex* **then**
    Convert a DFS tree rooted at t_i into a tree T_s rooted at s;
    return $\mathcal{P} \leftarrow$ `3-DPC_in_TCUBE`$(T_s, \{s\}, \{t_1, t_2, t_3\})$;
  $H \leftarrow$ the subgraph of G induced by $\{t_1, t_2, t_3\}$;
  switch $|E(H)|$ **do**
    case 1 **do** `//` `Assume` $E(H) = \{(t_1, t_2)\}$.
      Convert a spanning tree that contains (t_1, t_2) into a tree T_s;
      return $\mathcal{P} \leftarrow$ `3-DPC_in_TCUBE`$(T_s, \{s\}, \{t_1, t_2, t_3\})$;
    case 2 **do** `//` `Assume` $E(H) = \{(t_1, t_2), (t_2, t_3)\}$.
      if *an edge (t_2, t_j) joining sinks is not a bridge* **then**
        return $\mathcal{P} \leftarrow$ `3-DPC_in_GCUBE`$(G - (t_2, t_j), \{s\}, \{t_1, t_2, t_3\})$;
      return $\mathcal{P} \leftarrow$ `BuildDPC`$(G, t_2, \{s\}, \{t_1, t_2, t_3\})$; `// Fig. 3(a)`
    case 3 **do**
      Let (t_i, t_j) be the edge chosen in the proof of Theorem 5;
      return $\mathcal{P} \leftarrow$ `3-DPC_in_GCUBE`$(G - (t_i, t_j), \{s\}, \{t_1, t_2, t_3\})$;
    case 0 **do**
      if *there is no vertex u such that $\{t_1, t_2, t_3\} \subseteq N[u]$* **then**
        Build a spanning tree T_s rooted at s;
        return $\mathcal{P} \leftarrow$ `3-DPC_in_TCUBE`$(T_s, \{s\}, \{t_1, t_2, t_3\})$;
      Pick up a vertex u such that $\{t_1, t_2, t_3\} \subseteq N[u]$;
      if *an edge (u, t_i) joining u and a sink is not a bridge* **then**
        Convert a DFS tree T_{t_i} of $G - (u, t_i)$ rooted at t_i into T_s;
        return $\mathcal{P} \leftarrow$ `3-DPC_in_TCUBE`$(T_s, \{s\}, \{t_1, t_2, t_3\})$;
      return $\mathcal{P} \leftarrow$ `BuildDPC`$(G, u, \{s\}, \{t_1, t_2, t_3\})$; `// Fig. 3(b)`
  end
end

Function `BuildDPC`$(G, x, \{s\}, \{t_1, t_2, t_3\})$
  if $s \in N[x]$ *or* $v \in N[x]$ *for some v whose degree is* 1 **then**
    Build a spanning tree containing the edges of H and (s, x), if any;
    Convert the spanning tree into a tree T_s rooted at s;
    return $\mathcal{P} \leftarrow$ `3-DPC_in_TCUBE`$(T_s, \{s\}, \{t_1, t_2, t_3\})$;
  $(T_x, z) \leftarrow$ `DFS`(G, x); `//` (z, x) `is a back edge; assume` $s \notin V(T_{t_1})$
  $P \leftarrow$ `HP_in_TCUBE`(T_{t_1}, w, t_1) for a child w of t_1;
  Convert $T_x - T_{t_1}$ into a tree T'_s rooted at s;
  if $z \neq s$ **then**
    $\mathcal{P}' \leftarrow$ `3-DPC_in_TCUBE`$(T'_s, \{s\}, \{z, t_2, t_3\})$;
    return $\mathcal{P} \leftarrow (\mathcal{P}' - P') \cup \{P' \circ P\}$, where P' is the s–z path in $\mathcal{P}'$;
  else
    $P' \leftarrow$ `HP_in_TCUBE`(T'_s, t_2, t_3);
    Transform P' into a 2-DPC $\mathcal{P}'$ joining $\{s\}$ and $\{t_2, t_3\}$;
    return $\mathcal{P} \leftarrow \mathcal{P}' \cup \{\langle s \rangle \circ P\}$;
end

Remark. For given terminal sets $S = \{s\}$ and $T = \{t\}$ in G, a one-to-one 3-DPC of G^3 joining S and T can also be built in linear time using `3-DPC_in_GCUBE()`. According to the proof of Theorem 2 in [12], it is possible to find two vertices t_2 and t_3 with $(t_2, t), (t_3, t) \in E(G^3)$ in linear time such that G^3 admits a one-to-many 3-DPC joining $\{s\}$ and $\{t, t_2, t_3\}$. Then, it simply remains to extend its s–t_2 and s–t_3 paths to t, respectively, to obtain a desired one-to-one 3-DPC.

ACKNOWLEDGEMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (Nos. 2015R1D1A1A09056849 and 2017R1D1A1B03029625).

References

[1] K. Asdre, S. D. Nikolopoulos, The 1-fixed-endpoint path cover problem is polynomial on interval graphs, Algorithmica 58 (3) (2010) 679–710.
[2] M.-S. Chang, M.-T. Ko, H.-I. Lu, Linear-time algorithms for tree root problems, Algorithmica 71 (2) (2015) 471–495.
[3] X.-B. Chen, Paired 2-disjoint path covers of multidimensional torus networks with faulty edges, Information Processing Letters 116 (2) (2016) 107–110.
[4] T. Dvořák, P. Gregor, V. Koubek, Generalized Gray codes with prescribed ends, Theoretical Computer Science 668 (2017) 70–94.
[5] J. E. Hopcroft, R. E. Tarjan, Efficient algorithms for graph manipulation (Algorithm 447), Commun. ACM 16 (6) (1973) 372–378.
[6] I. Ihm, J.-H. Park, A linear-time algorithm for finding a paired 2-disjoint path cover in the cube of a connected graph, Discrete Applied Mathematics 218 (2017) 98–112.
[7] J. J. Karaganis, On the cube of a graph, Canad. Math. Bull 11 (1968) 295–296.
[8] L. C. Lau, D. G. Corneil, Recognizing powers of proper interval, split, and chordal graphs, SIAM Journal on Discrete Mathematics 18 (1) (2004) 83–102.
[9] Y.-L. Lin, S. S. Skiena, Algorithms for square roots of graphs, SIAM Journal on Discrete Mathematics 8 (1) (1995) 99–118.
[10] R. Motwani, M. Sudan, Computing roots of graphs is hard, Discrete Applied Mathematics 54 (1) (1994) 81–88.
[11] S. C. Ntafos, S. L. Hakimi, On path cover problems in digraphs and applications to program testing, IEEE Transactions on Software Engineering 5 (5) (1979) 520–529.
[12] J.-H. Park, I. Ihm, Single-source three-disjoint path covers in cubes of connected graphs, Information Processing Letters 113 (14–16) (2013) 527–532.
[13] J.-H. Park, I. Ihm, Disjoint path covers in cubes of connected graphs, Discrete Mathematics 325 (2014) 65–73.
[14] J.-H. Park, H.-C. Kim, 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) (2006) 227–240.
[15] J.-H. Park, H.-C. Kim, H.-S. Lim, Many-to-many disjoint path covers in the presence of faulty elements, IEEE Transactions on Computers 58 (4) (2009) 528–540.
[16] 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 (1960) 137–142.