社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－AL－118 (5)
2008／5／27

有向パスグラフの最大連結支配集合分割

水戸 将弥[1]　藤田 聡[2]

1 ERATO前中センシング融合プロジェクト
〒671-2280 姫路市書写2167 兵庫県立大学内
2 広島大学 大学院工学研究科 情報工学専攻
〒739-8527 東広島市鏡山1-4-1

概要

本稿では，グラフの最大連結支配集合分割問題について考察する．以下では，有向パスグラフの部分クラスに対してこの問題が多項式時間で解けることを示す．有向パスグラフとは，有向木上の有向パスによってモデル化される交差グラフであり，区間グラフのひとつの拡張になっている．本稿で考える有向パスグラフの部分クラスは，有向木中の入次数2以上のノードが高々ひとつであるという制約が課せられたクラスである．

Maximum Connected Domatic Partition of Directed Path Graphs

Masaya MITO[1]　Satoshi FUJITA[2]

1 JST ERATO Maenaka Human-Sensing Fusion Project
Hyogo Prefectural Unversity, Shosha 2167, Mimeji, 671-2280 Japan
2 Department of Information Engineering, Graduate School of Engineering
Hiroshima University, Kagamiyama 1-4-1, Higashi-Hiroshima, 739-8527 Japan

Abstract

In this paper, we consider the problem of finding a maximum connected domatic partition of a given graph. We propose a polynomial time algorithm for solving the problem for a subclass of directed path graphs which is known as a class of intersection graphs modeled by a set of directed paths on a directed tree. More specifically, we restrict the class of directed path graphs in such a way that the underlying directed tree has at most one node to have several incoming arcs.

1 Introduction

A connected dominating set (CDS, for short) for graph G is a dominating set which induces a connected subgraph of G [8, 13]. A **connected domatic partition** (CDP, for short) of G is a partition of the vertex set of G such that each subset in the partition is a CDS for G. In the literature, it is pointed out that CDS plays an important role in the resource allocation in computer networks, such as the message routing in wireless ad hoc networks [5, 14, 15], collective communication in sensor networks [3, 6, 9], and so on.

Let $d_c(G)$ denote the cardinality of a largest CDP of graph G [11, 12]. The problem of finding a CDP of maximum cardinality is known to be NP-hard for general graphs [6], and there have been derived several interesting results on the bound of value $d_c(G)$; e.g., it satisfies $d_c(G) \leq \kappa(G)$ unless G is a complete graph [16], where $\kappa(G)$ denotes the vertex connectivity of graph G, and it satisfies $d_c(G) \leq 4$ for any planar graph G [10]. It is also known that the above maximization problem can be solved in polynomial time for several classes of easy instances such as trees, cycles, and complete bipartite graphs. Unfortunately however, unlike ordinary domatic partition

problem which has been investigated during these decades [1, 2], very few is known about the "connected" version of the partitioning problem.

In this paper, we first point out that the problem of finding a maximum CDP can be solved in linear time for the class of interval graphs. We then extend the result to a subclass of directed path graphs, which is an intersection graph modeled by a set of directed paths on a directed tree with a single "joining" node characterized by a vertex set called junction (a formal definition of the class of considered graphs will be given in Section 4). The basic idea of the proposed scheme is to focus on a set of *critical* vertices in a junction, and to carefully partition such vertex set by solving the *k-edge-coloring problem* for a corresponding bipartite multigraph. An extension of the greedy partitioning scheme used for the class of interval graphs could be effectively applied to complete the partitioning of the remaining vertices.

The remainder of this paper is organized as follows. Section 2 introduces necessary definitions. Section 3 describes a linear time algorithm for solving the partitioning problem for directed path graphs modeled by a directed tree with a single source. An extension of the algorithm to the case with a single junction is given in Section 4. Finally, we conclude the paper with future problems in Section 5.

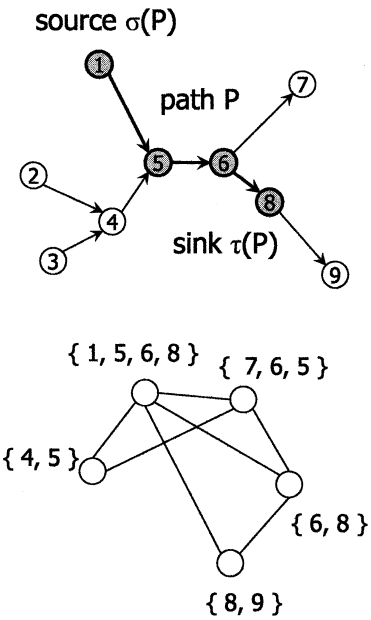


Figure 1: An example of DPG (the upper figure is the underlying directed tree, and the lower figure is the corresponding DPG).

2 Preliminaries

Let T be a directed tree consisting of node set W and arc set A. If there is a directed path from node x to node y in T, then we say that they satisfy relation $x < y$, and that x is an ancestor of y or y is a descendant of x. A node with no descendant is called a sink, and a node with no ancestor is called a source. Given directed path P in T, the first and the last nodes of the path are denoted as $\sigma(P)$ and $\tau(P)$, respectively. (For brevity, we will use a similar notation for the set of paths.)

Graph $G = (V, E)$ is called a **directed path graph** (DPG, for short) if it is an intersection graph modeled by a set of directed paths in a directed tree T, i.e., V corresponds to a set of directed paths in T and two vertices[1] in V are connected by an edge iff their corresponding paths share at least one node (see Figure 1 for illustration). In the following, we say that "G is modeled by T" if the meaning of the sentence is clear from the context. In addition, we identify a vertex v in G with its corresponding directed path in T. For example, we often say that vertex v "contains" node x if the corresponding path contains x. For any subset V' of V, we use symbol V' to denote a subgraph of T which is obtained by taking a union of paths corresponding to the vertices in V'. Moreover, for any subgraph T' of T and a vertex set $V' \subseteq V$, a subgraph of T, which is obtained by taking an intersection of T' and V', is denoted by $T' \cap V'$.

[1]Throughout this paper, we will use terms "vertex" and "edge" for graph G, and distinguish them from terms "node" and "arc" used for tree T.

3 Tree with Single Source

In this section, we propose a simple algorithm to find an optimal CDP of DPG modeled by a tree with a single source. Note that this algorithm will be used as a subroutine in the next section.

3.1 Interval Graphs

To clarify the basic idea of the scheme, we first consider the case in which given G is modeled by a directed tree with a single source s and a single sink t; i.e., when G is an interval graph. The proposed scheme partitions V into $\kappa(G)$ CDSs for G; i.e., it

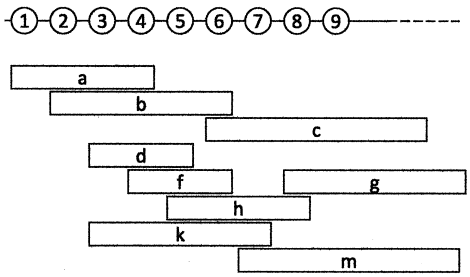


Figure 2: An interval graph (the upper figure is a path of nodes and the lower figure shows an interval model of the given interval graph which is a collection of paths).

is optimal since $\kappa(G)$ is an upper bound on the cardinality of CDP. Let $\mathcal{C}$ be a set of $\kappa(G)$ subsets of vertices, each of which is intended to represent a CDS for G. The proposed algorithm proceeds as follows:

- Let $U \leftarrow V$ and initialize each element in $\mathcal{C}$ to $\emptyset$.
- Repeat the following operations until every element in $\mathcal{C}$ becomes a CDS for G.
  - Let c^* be an element in $\mathcal{C}$ such that $\tau(c^*) = \min_{c \in \mathcal{C}}\{\tau(c)\}$, where $\tau(c)$ denotes the last node of path c which is defined as $\min_{v \in U}\{\sigma(v)\}$ if c is an empty set.
  - Let v be a vertex in U such that $\sigma(v) \leq \tau(c^*) < \tau(v)$. If U contains no such vertex, then output "failed" and terminate.
  - Move vertex v from U to c^*.
- Output $\mathcal{C}$ as a solution after adding the remaining vertices in U (if any) to an arbitrary element in $\mathcal{C}$, then terminate.

Figure 2 illustrates a part of interval graphs, and we will explain the behavior of the above algorithm by using this graph. Assume that the vertex connectivity of the graph is three, i.e., we will partition the vertex set into three subsets C_1, C_2, and C_3. After initializing set U to V, it picks up vertex a (which corresponds to a path in the interval model) as the first element in C_1, since it has the leftmost source node 1. It then picks up vertices b and d as the first elements of C_2 and C_3, respectively. After that, we will focus on subset C_1 and try to extend it to the right hand side, since it has the leftmost terminal node at this time. In fact, we can find a vertex which enables such extension (e.g., vertex f or k), and could proceed a similar operation until all subsets become a (connected) dominating set for the given graph.

Proposition 1 *If G is a DPG modeled by a directed path, then the above algorithm outputs a CDP of cardinality $\kappa(G)$ in linear time.*

Proof. Since the time complexity is obvious by the description, in the following, we show that it always outputs a required partition. Let c be an element in $\mathcal{C}$ selected at the beginning of an iteration, and without loss of generality, let us assume that there is an arc a to have $\tau(c)$ as its predecessor (since otherwise, c has already been a CDS for G). Let α_a denote the number of paths containing arc a. By the description of the algorithm, if $\mathcal{C}$ contains an element (i.e., a path) which does not contain arc a, then any element (i.e., path) containing the arc can never be selected as the candidate. Thus, since $\kappa(G) = \min_{a \in A}\{\alpha_a\}$, the scheme associates $\kappa(G)$ vertices containing arc a to different elements in $\mathcal{C}$, in such a way that every element contains exactly one such vertex. A similar claim holds for every arc in the given T. Thus the proposition follows. Q.E.D.

3.2 A Generalization

The above idea can be easily generalized to trees with a single source and several sinks. Let T be a directed tree with a single source s and ℓ sinks $t_1, t_2, \ldots, t_\ell$. Let P_j denote the path from s to t_j, and let $T_0, T_1, \ldots, T_{\ell-1}$ be subtrees of T defined as: $T_0 \stackrel{\text{def}}{=} \{s\}$, and $T_j \stackrel{\text{def}}{=} T_{j-1} \cup P_j$ for each $1 \leq j \leq \ell - 1$. Let $\hat{P}_j$ be the path from node $\tau(P_j \cap T_{j-1})$ to node t_j. Note that T_j can also be represented as $T_0 \cup (\bigcup_{i=1}^{j} \hat{P}_i)$.

A generalized algorithm for such tree T proceeds as follows:

- Let $U \leftarrow V$ and initialize each element in $\mathcal{C}$ to $\emptyset$.
- For $j = 1$ to ℓ sequentially, repeat the following operation until every element in $\mathcal{C}$ contains a vertex containing sink t_j:
  - Let c^* be an element in $\mathcal{C}$ such that $\tau(c^* \cap \hat{P}_j) = \min_{c \in \mathcal{C}}\{\tau(c \cap \hat{P}_j)\}$, where $\tau(c^* \cap \hat{P}_j)$ is defined as the first node of $\hat{P}_j$ if $c^* \cap \hat{P}_j = \emptyset$.
  - Let v be a vertex in U such that $\sigma(v \cap \hat{P}_j) \leq \tau(c^* \cap \hat{P}_j) < \tau(v \cap \hat{P}_j)$. If U contains

no such vertex, then output "failed" and terminate.

- Move the vertex v from U to c^*.

- Output $\mathcal{C}$ as a solution after adding the remaining vertices in U (if any) to an element in $\mathcal{C}$, then terminate.

Proposition 2 *If G is a DPG modeled by a directed tree with a single source, the above algorithm outputs a CDP of size $\kappa(G)$ in linear time.*

Proof. The case of $\ell = 1$ is immediate from Proposition 1. To prove the claim for $\ell \geq 2$, it is enough to show that the following three conditions hold at the beginning of the j^{th} iteration for each $2 \leq j \leq \ell$: 1) every element in $\mathcal{C}$ contains a vertex containing $\sigma(\hat{P}_j)$, 2) if there is an element in $\mathcal{C}$ which contains a vertex v containing an arc in $\hat{P}_j$, then the vertex v must be reachable from $\sigma(\hat{P}_j)$ in $\mathcal{C}$, and 3) for any arc a in $\hat{P}_j$, each element in $\mathcal{C}$ contains at most one vertex containing a.

The first condition is immediate since every element in $\mathcal{C}$ is a CDS for a subgraph of G modeled by T_{j-1} at the beginning of the j^{th} iteration. The second condition can be verified by considering a vertex v such that $\sigma(v \cap \hat{P}_j) > \sigma(\hat{P}_j)$, which is never being selected during the processing for sinks $t_1, \ldots, t_{j-1}$, since $T_{j-1} \cap v = \emptyset$. Finally, the third condition can be proved by using an argument similar to the proof of Proposition 1. Hence, the proposition follows. Q.E.D.

4 Graph with Single Junction

4.1 Definitions

In the following, we let $k = \kappa(G)$, for brevity. In this section, we consider a class of trees which contains exactly one node to have more than one incoming arcs. Note that it is a generalization of the class of trees considered in the last section, since it allows the existence of several sources, as long as the number of "joins" is restricted to one.

Let J be the set of all vertices containing such a joining node w, which will be referred to as the **junction** with respect to w. Note that a junction induces a clique in the given graph, and that it partitions the given vertex set into three subsets as J, $V^+ \stackrel{\text{def}}{=} \{v \mid w < \sigma(v)\}$, and $V^- \stackrel{\text{def}}{=} V \setminus (J \cup V^+)$. Figure 4 illustrates a junction. For each node x in T, let $f(x)$ denote the number of paths in V containing both w and x. A **lower boundary** of a subtree of T centered at node w, is defined as follows:

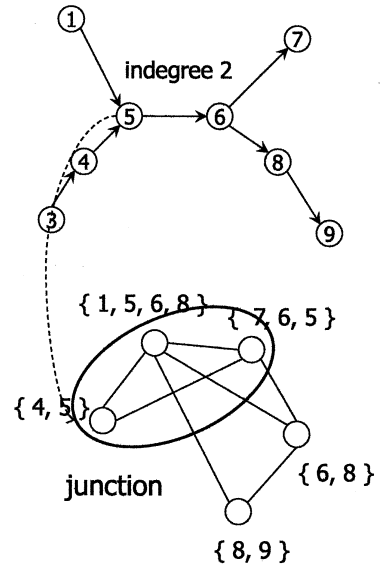
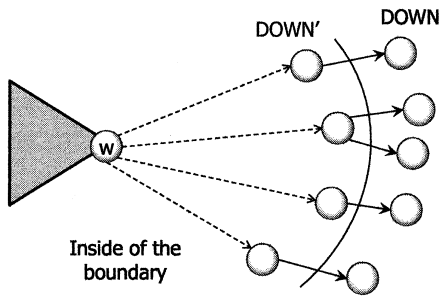


Figure 3: Junction (note that a junction is a set of vertices in V which forms a clique in the given DPG).

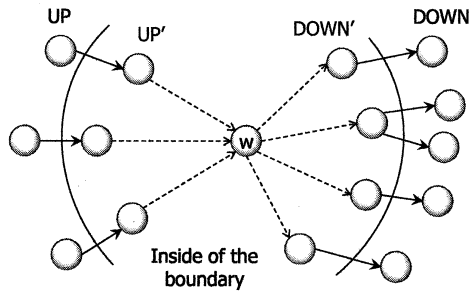
Definition 1 (Lower Boundary) Let W^+ be the set of descendants x of w such that: 1) $f(x) \leq k$ and 2) a predecessor y of x satisfies $f(y) > k$. Given W^+, lower boundary W_L is defined as the set of nodes x satisfying either: 1) a successor of x is in W^+, or 2) x is a sink with $f(x) > k$.

A vertex in junction J is said to be **critical** if it corresponds to a path in the tree containing a node in the lower boundary W_L. Let J^* be the set of critical vertices. For each node x in T, let $f^*(x)$ denote the number of vertices in J^* containing x. With the above notions, an upper boundary of the subtree with respect to node w is defined as follows:

Definition 2 (Upper Boundary) If $f^*(w) = k$, upper boundary W_U and set W^- are both defined as $\{w\}$. Otherwise, W^- is the set of ancestors x of w such that: 1) $f^*(x) \leq k$ and 2) a successor y of x satisfies $f^*(y) > k$; and given set W^-, upper boundary W_U is defined as the set of nodes x satisfying either: 1) a predecessor of x is in W^-, or 2) x is a source with $f^*(x) > k$.



(a) Lower boundary (DOWN stands for W^+ and DOWN' stands for W_L).



(b) Upper boundary (UP stands for W^- and UP' stands for W_U).

Figure 4: Upper and lower boundaries of critical vertices around junction centered at node w.

4.2 Basic Strategy for Partitioning Critical Vertices

The proposed scheme tries to find a partition $\mathcal{C}$ of J^* to satisfy the following two conditions:

- SHARE: For any $c \in \mathcal{C}$ and for any $x \in W^- \cup W^+$, $J^* \cap c$ contains at most one path containing node x.
- RESERVE: For any $c \in \mathcal{C}$ and for any $x \in W_U \cup W_L$, $J^* \cap c$ contains at least one path containing node x.

In this and the next subsection, we describe how to find such a partitioning in polynomial time, and in Subsection 4.4, we explain how those conditions are related with the overall partitioning of V. The basic idea for the partitioning of J^* to satisfy the above two conditions is to use k-edge-coloring of a bipartite multigraph reflecting the structure of critical vertices. More precisely, we consider a bipartite multigraph H with vertex set $V_H = W^- \cup W^+$ and edge set $E_H (\subset W^- \times W^+)$, where vertices x and y are connected by an edge in E_H iff J^* contains a path containing both x and y. Note that there may exist vertices in J^* which have no corresponding edge in E_H; e.g., a path terminating at the lower boundary does not contain a node in W^+ (handling of such "hidden" vertices will be discussed in the next subsection).

Let ϕ be a k-edge-coloring of H; i.e., ϕ is a function from E_H to $\{1, 2, \ldots, k\}$ such that any two adjacent edges are assigned different colors. Note that graph H is k-edge-colorable in $O(|E_H| \log k)$ time, since the maximum degree of H is at most k [4]. Given a k-edge-coloring ϕ of H, let us consider the following natural association of vertices in J^* to the elements in $\mathcal{C}$:

1) If an edge in E_H corresponding to vertex $v (\in J^*)$ is assigned color i by ϕ, then v is associated to the i^{th} element in $\mathcal{C}$.

2) Otherwise, the vertex is associated to an element in $\mathcal{C}$ which contains no vertex sharing the same node in $W^- \cup W^+$ with v. Note that such element always exists in $\mathcal{C}$, since $f^*(x) \leq k$ for any $x \in W^- \cup W^+$.

The above assignment obviously satisfies condition SHARE. However, the second condition RESERVE cannot always be satisfied if we directly apply the procedure to the original H (due to the problem of "hidden" vertices). In the next subsection, we show that the second condition can always be satisfied if we conduct an appropriate preprocessing for the modification of graph H before conducting the above assignment procedure.

4.3 Preprocessing to Satisfy Condition RESERVE

In the preprocessing phase, bipartite multigraph H is modified for each node in W_L, according to the type of node defined as follows: A node of Type 1 is a sink; a node of Type 2 has a successor y with $f^*(y) \geq k$; and a node of Type 3 or 4 has a successor but none of them has a f^* value greater than or equal to k. The difference of the last two types is the summation of f^* values over all successors; i.e., in Type 3, the summation is bounded by k, but it is greater than k for Type 4. Recall that the objective of the preprocessing phase is to satisfy RESERVE after simply applying the above assignment procedure.

Type 1: Recall that a sink $x \in W_L$ is not contained in the vertex set of the original H. Let

$S(\subseteq J^*)$ be a vertex set consisting of arbitrary k paths containing sink x. Add a new vertex corresponding to x to V_H, and connect it to k nodes in W^- via edges corresponding to the selected k paths.

Type 2: No modification is necessary if x has a successor y with $f^*(y) \geq k$. In fact, when $f^*(y) = k$, since it associates exactly one path containing y to c for any $c \in \mathcal{C}$, condition RESERVE obviously holds for its parent x. When $f^*(y) > k$, on the other hand, since x must have a descendant of the other type contained in W_L, the condition holds for x as long as it is satisfied for at least one of its descendants.

Type 3: Let S' be the set of successors of x, and $f^*(S') \stackrel{\text{def}}{=} \sum_{y \in S'} f^*(y)$. If $f^*(S') \leq k$, then contract vertices in $V_H \cap S'$ to a single vertex $\hat{x}$, i.e., $V_H \leftarrow (V_H \setminus S') \cup \{\hat{x}\}$, and connect it to nodes in W^- via corresponding edges. Then, after selecting arbitrary $f^*(x) - f^*(S')$ (> 0) paths terminating at node x from J^*, add edges corresponding to the selected paths to E_H. Note that this modification obviously satisfies RESERVE for node x, and does not violate SHARE for any $y \in S'$.

Type 4: Let $S' = \{y_1, y_2, \ldots, y_\ell\}$ be the set of successors of x. At first, for each $y_j \in V_H \cap S'$, add new vertex $\hat{y}_j$ to V_H, and connect it with y_j via $k - f^*(y_j)$ (> 0) parallel edges. Then, after adding another vertex p to V_H, connect it with vertices in $\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_\ell\}$ in the following manner: Let i be an integer such that $\sum_{j=1}^{i-1} f^*(y_j) < k$ and $\sum_{j=1}^{i} f^*(y_j) \geq k$. Vertex p is connected with $\hat{y}_j$ via $f^*(y_j)$ parallel edges for $1 \leq j \leq i-1$, and connected with vertex $\hat{y}_i$ via $k - \sum_{j=1}^{i} f^*(y_j)$ parallel edges. Note that in any k-edge-coloring of the resultant H, k edges incident on p are assigned distinct colors, which will be propagated to edges incident on S' via vertices in $\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_\ell\}$. Thus, it satisfies condition RESERVE for node x.

A similar modification could be done for each node in W_U. Hence, a proof of the satisfaction of RESERVE will complete by proving the following lemma.

Lemma 1 *In the resultant graph H, each vertex in J^* corresponds to at most one edge in E_H.*

Proof. The claim apparently holds for the original H. An addition of edges corresponding to vertices in J^* takes place only when it examines node x of Type 1 or 3. In the former case, the path from w to x never contains a node in W^+, and even if it contains a node in W_L, the modification of H for the node in W_L does not take place since it should be a node of Type 2. When x is a node of Type 3, on the other hand, every edge added by the modification corresponds to a path terminating at node x. Thus, it does not contain a node in W^+, and even if it contains a node in W_L, it does not cause a modification of H. A similar argument holds for the upper boundary W_U. Hence, the lemma follows. Q.E.D.

4.4 Partition of the Remaining Vertices

Now, we have obtained a partition $\mathcal{C}$ of J^* satisfying conditions SHARE and RESERVE. Given such partition $\mathcal{C}$, a greedy assignment scheme described in Section 3 correctly finds a (connected domatic) partition of $J^* \cup V^+$ in linear time, where $V^+ = \{v \mid w < \sigma(v)\}$. Thus, in the remaining of this section, we describe how to realize a correct partition of $V^- = V \setminus (J^* \cup V^+)$.

For each source s in T, let Z_s denote the set of sinks reachable from s without passing through node w; let T_s denote an out-tree which is obtained by taking a union of paths connecting from s to nodes in Z_s, and let P_s denote the unique path from s to w. The procedure for the partition of V^- proceeds as follows: First, for each source s, it assigns paths on P_s to $\mathcal{C}$ with an algorithm described below. After that, it assigns paths in T_s to $\mathcal{C}$ by using a greedy scheme for out-trees described in Section 3.2.

More concretely, the partition of paths on P_s proceeds as follows:

- Let $U \leftarrow V^-$, initialize each element in $\mathcal{C}$ to an empty set, and initialize $\mathcal{D}$ to the output of previous procedures; i.e., $\mathcal{D}$ is a (connected domatic) partition of $J^* \cup V^+$ of size k.
- Repeat the following operation until every element in $\mathcal{C}$ contains each arc on P_s.
  - Let c^* be an element in $\mathcal{C}$ such that $\tau(c^* \cap P_s) = \min_{c \in \mathcal{C}}\{\tau(c \cap P_s)\}$, where $\tau(c^* \cap P_s)$ is defined as source s if $c = \emptyset$. Let d^* be an element in $\mathcal{D}$ such that $\sigma(d^* \cap P_s) = \min_{d \in \mathcal{D}}\{\sigma(d \cap P_s)\}$.
  - Let v be a vertex in U such that $\sigma(v \cap P_s) \leq \tau(c^* \cap P_s) < \tau(v \cap P_s)$. If U contains no such vertex and $\tau(c^* \cap P_s) < \sigma(d^* \cap P_s)$, then output "failed" and terminate.
  - Move vertex v from U to c^* if U contains v. Otherwise, $c^* \leftarrow c^* \cup d^*$ and $\mathcal{D} \leftarrow \mathcal{D} \setminus \{d^*\}$.
- Output $\mathcal{C}$ as a solution, and terminate.

Proposition 3 *If $\mathcal{D}$ is initialized to the output of previous procedures, then the above procedure successfully outputs necessary partition in linear time.*

Proof. In the following, we will merely consider the correctness of scheme, since the time complexity is obvious by description. Let c be an element in $\mathcal{C}$ selected at the beginning of an iteration, and suppose that $\tau(c \cap P_s) < w$, without loss of generality. (Note that if $\tau(c \cap P_s) = w$, c has already contained every arc on P_s.) Let a be an arc on P_s starting from node $\tau(c \cap P_{s,w})$, and let α_a denote the number of vertices in G containing arc a. By the description of the algorithm, if there is an element in $\mathcal{C}$ which does not contain a, then any element in $\mathcal{C}$ containing arc a can never be selected as the candidate. In addition, since $\mathcal{D}$ is initialized to a partition of $J^* \cup V^+$ satisfying SHARE and RESERVE for nodes in $(W^- \cup W_U) \cap P_s$, it does not contain an element which contains more than one vertices containing arc a, or every element in $\mathcal{D}$ has contained arc a. Thus, since the vertex connectivity $k(= \kappa(G))$ is represented as $\min_{a \in A}\{\alpha_a\}$, element c can contain a vertex containing arc a. A similar claim holds for every arc on path P_s. Thus, the proposition follows. Q.E.D.

Finally, we can easily show that the greedy scheme given in Section 3.2 works well even for out-tree T_s. Thus, we have the following proposition.

Proposition 4 *If G is a DPG modeled by a directed tree containing at most one node to have several incoming arcs, then the connected domatic partition problem can be solved in polynomial time.*

5 Concluding Remarks

This paper proposed an algorithm for solving the connected domatic partition problem for a subclass of directed path trees. An important open problem is to examine if the proposed algorithm can be extended to general directed path graphs with more than one junctions. We have a positive conjecture for this problem, such that it could be solved by repeatedly applying the proposed assignment procedure for each junction (probably, it needs an appropriate ordering of such junctions). Since it can be easily shown that an undirected version of the problem is NP-hard using a reduction from the 3-edge-coloring problem [7], it would clarify a sharp boundary on the complexity of the connected domatic partition problem.

References

[1] A. A. Bertossi. On the domatic number of interval graphs. *Information Processing Letters*, 28(6):275–280, August 1988.

[2] M. A. Bonuccelli. Dominating sets and domatic number of circular arc graphs. *Discrete Applied Mathematics*, 12:203–213, 1985.

[3] M. Cardei and D.-Z. Du. Improving Wireless Sensor Network Lifetime through Power Aware Organization. *ACM Wireless Networks*, 11(3):333–340, 2005.

[4] R. Cole, K. Ost and S. Schirra. Edge-Coloring Bipartite Multigraphs in $O(E \log D)$ Time. *Combinatorica*, 21(1):5–12, 2001.

[5] F. Dai and J. Wu. An Extended Localized Algorithm for Connected Dominating Set Formation in Ad Hoc Wireless Networks. *IEEE Transactions on Parallel and Distributed Systems*, 53(10):1343–1354, 2004.

[6] Q. Dong. Maximizing System Lifetime in Wireless Sensor Networks. In *Proc. of the 4th International Symposium on Information Processing in Sensor Networks*, 13–19, 2005.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, 1979.

[8] S. Guha and S. Khuller. Approximation Algorithms for Connected Dominating Sets. In *Proc. European Symposium on Algorithms*, 179–193, 1996.

[9] R. W. Ha, P. H. Ho, X. Shen and J. Zhang. Sleep Scheduling for Wireless Sensor Networks via Network Flow Model. *Computer Communications*, 29(13-14):2469-2481, 2006.

[10] B. L. Hartnell and D. F. Rall. Connected Domatic Number in Planar Graphs. *Czechoslovak Mathematical Journal*, 51(1):173–179, 2001.

[11] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of Domination in Graphs.* Marcel Dekker, Inc., 1998.

[12] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Domination in Graphs: Advanced Topics.* Marcel Dekker, Inc., 1998.

[13] S. Hedetniemi and R. Laskar. Connected domination in graphs. In *Graph Theory and Combinatorics*, Academic Press, London, pp. 209-218, 1984.

[14] J. Wu and H. Li. Domination and Its Applications in Ad Hoc Wireless Networks with Unidirectional Links. In *Proc. of International Conference on Parallel Processing*, pages 189–200, 2000.

[15] J. Wu. Extended Dominating-Set-Based Routing in Ad Hoc Wireless Networks with Unidirectional Links. *IEEE Transactions on Parallel and Distributed Computing*, 22:327–340, 2002.

[16] B. Zelinka. Connected Domatic Number of a Graph. *Math. Slovaca*, 36:387–392, 1986.