修士論文

幅の小さなDAGパス分解の研究
及びDAG木分解の提案

指導教員　　川原 純 准教授

京都大学大学院情報学研究科
修士課程通信情報システム専攻

伊豆 真哉

2025年2月1日

幅の小さなDAGパス分解の研究及びDAG木幅の提案

伊豆 真哉

内容梗概

有向有向グラフがどれだけ有向パスに近い構造をしているかを表すパラメータとしてDAGパス幅が存在する．DAGパス幅は有向パス幅のルールに対し，任意の枝について両端点を同時に含むバッグが存在するというルールを追加している．DAGパス幅は，有向パス幅が常に0になってしまうDAGに対しても非自明な幅を与えることができ，DAG上でNP困難な問題に対するパラメータ化アルゴリズムの構築に役立っている．またDAGパス幅に対する近似アルゴリズムは現在まで知られていなかった．

今回の研究では，まず最初にDAG上でもNP困難である様々な問題に対し，DAGパス幅を使ったパラメータ化アルゴリズムを提案する．具体的にはDirected Dominating Set Problem, Max Leaf Outbranching Problem, Directed Steiner Tree Problemの3つの問題に対しては，幅がwであるDAGパス分解が与えられたときに$f(w)\cdot g(n)$で厳密解を与えるFPTアルゴリズムを提案する．k-Disjoint Path Problemに対しては$O(k^w \cdot f(w)\cdot g(n))$で厳密解を与えるパラメータ化アルゴリズムを提案する．

次にDAGパス幅に対し，$O(\log^2 n)$の近似比をもつ多項式時間アルゴリズムを提案する．またOne Shot Black Pebbling ProblemがDAGパス分解を求める問題と等価であることを示し，これによって$O(\log^{3/2} n)$の近似比をもつDAGパス分解の近似アルゴリズムの存在を示す．

さらにDAGパス幅とグラフの最大出次数，根数がそれぞれw, d, lで表されるとき，$O(l\cdot d^w)$の幅を与えるアルゴリズムを提案する．このアルゴリズムは完全有向d分木のembeddingを行うことでアルゴリズムを構築している．

最後にDAGパス幅の一般化となる概念であるDAG木幅を新たに定義する．DAG木幅は有向グラフが有向木からどれだけ近いかを表すパラメータであり，DAG木幅の値が小さいほどグラフが有向木に近い構造をしていることを表す．DAG木幅を与える最適なDAG木分解の構成がNP困難であることを示し，その後DAG木幅をパラメータとしてDirected Dominating Set Problemを解く$f(w)\cdot g(n)$時間のFPTアルゴリズムを提案する．

A simple algorithm for finding DAG-path-decompositions of small width and proposal of DAG-treewidth

Shinya Izu

Abstract

abstruct

幅の小さなDAGパス分解の研究及びDAG木幅の提案

目次

第1章　はじめに

NP 困難な問題に対して高速に解を求める方法の一つとして木分解という手法が存在する．木分解とは，一般のグラフに対して全頂点をいくつかの頂点集合に分解し，それぞれを一つの節点と見なして木のような構造に変換する操作である．木分解を行うと，一般のグラフでも木に対するアルゴリズムが適応できるため，NP 困難な問題でも比較的高速に解を求めることができる．木分解を行ったとき，一つの節点に含まれる最大の頂点数を幅といい，あるグラフに対して全ての木分解を試したときの幅の最小値を木幅という．木幅が小さいほど木に近い構造をしていることを表す．この木幅が定数で抑えられる場合，NP 困難な問題に対しても頂点数の多項式時間で解くことができる場合がある．

木分解と同様に，グラフ上の頂点をいくつかの頂点集合に分解し，グラフ全体をパス型の構造に変換する操作をパス分解という．また一つの節点に含まれる最大の頂点数をパス幅という．こちらも一般のグラフをパスのように扱うことができるため，パス幅が定数で抑えられる場合，NP 困難な問題に対しても頂点数の多項式時間で解くことができる場合がある．

無向グラフに対するパス幅は，Robertson ら [1] によって 1983 年に初めて提案された．また，同氏らによって無向グラフに対する木幅 [2] も初めて提案されている．1987 年には，定数 k が与えられたとき，入力グラフの木幅やパス幅が k 以下かどうかを判定する問題が NP 完全であることが Arnborg ら [3] によって明らかにされている．1989 年には同氏らによって木幅やパス幅を用いた様々な FPT アルゴリズム [4] が研究されている．

木幅が k であるグラフが与えられたとき，幅が高々 k の定数倍である木分解を多項式時間で得るアルゴリズムが存在するかどうかはわかっていない．同様にパス幅についても定数近似が可能であるかはわかっていない．一方で木幅が k で抑えられる場合，木幅の $O(\sqrt{\log k})$ の近似比をもつ幅を与える多項式時間アルゴリズムの存在が Amir [5] によって示されている．また $2^{O(k)}$ 時間の 2-近似アルゴリズムが Tuukka [6] によって構築されている．同様にパス幅についても先程の k を用いて $O(k\sqrt{\log k})$ の近似比の多項式時間アルゴリズムが Carla ら [7] によって示されているほか，パス幅 pw を用いて $O(2^{pw})$ の幅を与える多項式時間アルゴリズムが Cattell ら [8] によって示されている．

有向グラフに対しても様々な幅が考えられてきた．1997 年に Reed [9] が有

向パス幅を提案しており，2001 年には Johnson ら [10] が有向木幅を提案している．また 2012 年には，Berwanger ら [11] によって DAG に対する幅が提案されている．

有向パス幅は有向グラフがどれだけ DAG に近い構造をしているかを表すパラメータである．そのため DAG に対しては有向パス幅は常に 0 になり，DAG 上の問題に対してパラメータ化アルゴリズムの構築が難しい場合があった．これに対し，2023 年に Kasahara ら [12] によって，新たに DAG パス幅が提案されている．DAG パス幅は有向有向グラフがどれだけ有向パスに近い構造をしているかを表すパラメータであり，有向パス分解の条件に加え，任意の枝に対してその端点を同時に含むバッグが存在することをルールに加えている．これにより有向グラフ上の強連結成分は同時に 1 つのバッグに含まれていなければならない．この条件によって特に DAG について非自明な幅が得られ，DAG に対して有効な FPT アルゴリズムの構築が可能となっている．[12] では DAG パス幅を用いた k-独立集合問題のパラメータ化アルゴリズムの構築や DAG パス幅を求めることが NP 困難であることの証明などが行われている．一方で k-独立集合問題以外での DAG 上でも NP 困難な問題に対するパラメータ化アルゴリズムの構築や DAG パス分解自体を求めるアルゴリズムは著者が知る限りこれまでわかっていなかった．

そこで今回は次の 3 つの研究を行った．1 つ目は DAG パス分解を用いて DAG 上での様々な NP 困難問題に対するパラメータ化アルゴリズムの構築である．2 つ目は DAG パス分解を求める近似アルゴリズム，及び制限された幅をもつ DAG パス分解の構築を行うパラメータ化アルゴリズムの提案である．3 つ目は DAG パス幅の拡張となる DAG 木幅の提案である．まず第 3 章では，DAG 上でも NP 困難な問題である Directed Dominating Set Problem, Max Leaf Outbranching Problem の 2 つの問題に対して，頂点数が n の DAG，幅が w の DAG パス分解を入力したとき，それぞれ $O(2^w wn), O(2^w wn + n^2)$ 時間の FPT アルゴリズムを提案する．Directed Steiner Tree Problem に対しては，ターミナル集合のサイズを k としたときに $O(2^w(k+w)n + n^2)$ 時間の FPT アルゴリズムを提案する．また k-Disjoint Path Problem に対しては $O((k+1)^w(w^2+k)n + n^2)$ 時間のパラメータ化アルゴリズムを提案する．本論文では DAG 上であることを前提としているが，一般の有向グラフに対しても強連結成分分解を行うことで本アルゴリズムを提供することができる．次に第 4 章では DAG パス幅に対し，$O(\log^2 n)$-

近似アルゴリズムを提案する．またより近似比の小さい$O(\log^{3/2} n)$-近似アルゴリズムの存在を示す．これは one-shot Black Pebbling Problem が DAG パス分解を求める問題と等価であることから示される．さらに第 5 章では，DAG パス幅とグラフの最大出次数，根数をそれぞれw, d, lと表したとき，$O(l \cdot d^w)$の幅を与えるアルゴリズムを提案する．このアルゴリズムは無向パス分解に対するアルゴリズム [8] を参考にして構築しており，完全有向d分木の embedding を行うことでアルゴリズムを構築している．最後に第 6 章, 第 7 章では DAG パス幅の一般化となる概念である DAG 木幅の提案を行っている．DAG 木幅は有向グラフが有向木からどれだけ近いかを表すパラメータであり，DAG 木幅の値が小さいほどグラフが有向木に近い構造をしていることを表す．第 6 章で DAG 木幅を与える最適な DAG 木分解の構成が NP 困難であることを示し，第 7 章で DAG 木幅をパラメータとして Directed Dominating Set Problem を解く$O(2^w wn)$時間の FPT アルゴリズムを提案している．

第2章　準備

2.1　DAG

本研究では入力グラフをDAGとしたものを多く扱う．DAGは以下のように定義される．

Definition 1. DAG (Directed Acyclic graph) とは閉路のない有向グラフのことである．

またDAGの各頂点に対して先行頂点集合と後続頂点集合を定める．

Definition 2. DAG $G = (V, E)$ のある頂点 $v \in V$ に対し，v の先行頂点集合 $\mathsf{pred}(v)$ と後続頂点集合 $\mathsf{suc}(v)$ を以下のように定める．

$$\mathsf{pred}(v) = \left\{ u \in V \;\middle|\; (u, v) \in E \right\}$$
$$\mathsf{suc}(v) = \left\{ w \in V \;\middle|\; (v, w) \in E \right\}$$

2.2　様々なパス分解とパス幅

本研究ではDAGパス幅についての研究であるが，DAGパス幅の特徴を理解しやすくするため，比較として (無向) パス幅，有向パス幅の定義を示す．まず無向グラフ上のパス分解・パス幅の定義を示す．

Definition 3 (パス分解)**.** [1] $G = (V, E)$ を無向グラフとする．G のパス分解 (undirected PD) とは，以下の3つの条件をみたすような V の部分集合 X_i $(i = 1, 2, \ldots, s)$ の列 $X = (X_1, X_2, \ldots, X_s)$ である．

1. $X_1 \cup X_2 \cup \cdots \cup X_s = V$
2. 任意の枝 $(u, v) \in E$ に対し，ある i (≥ 1) があり，$u, v \in X_i$
3. 任意の整数 i, j, k $(1 \leq i \leq j \leq k \leq s)$ について，$X_i \cap X_k \subseteq X_j$．すなわち任意の頂点 $v \in V$ について，v は X 上でただ一つの非空なパスを誘導する．

Definition 4 (パス幅)**.** 無向グラフ G のパス分解 $X = (X_1, X_2, \ldots, X_s)$ に対し，$\max_i\{|X_i| - 1\}$ を X の幅という．G のパス幅とは，G の全てのパス分解を考えたときの幅の最小値である．

パス幅はグラフがどれだけパスに近い構造をしているかを表すパラメータであり，パス幅の値が小さいほどパスに近い構造をしていることを表す．一般の無向グラフに対してパス幅を求める問題はNP困難である [3]．

次に有向パス分解・有向パス幅についての定義を示す．

Definition 5 (有向パス分解). [9] $G = (V, E)$ を有向グラフとする．G の有向パス分解 (directed PD) とは，以下の 3 つの条件をみたすような V の部分集合 X_i $(i = 1, 2, \ldots, s)$ の列 $X = (X_1, X_2, \ldots, X_s)$ である．

1. $X_1 \cup X_2 \cup \cdots \cup X_s = V$
2. 任意の有向枝 $(u, v) \in E$ に対し，ある i, j $(i \leq j)$ があり，$u \in X_i$, $v \in X_j$
3. 任意の i, j, k $(1 \leq i \leq j \leq k \leq s)$ について，$X_i \cap X_k \subseteq X_j$．すなわち任意の頂点 $v \in V$ について，v は X 上でただ一つの非空なパスを誘導する．

Definition 6 (有向パス幅). 有向グラフ G の有向パス分解 $X = (X_1, X_2, \ldots, X_s)$ に対し，$\max_i\{|X_i| - 1\}$ を X の幅という．G の有向パス幅とは，G の全ての有向パス分解を考えたときの幅の最小値である．

有向パス幅はグラフがどれだけ DAG に近い構造をしているかを表すパラメータであり，有向パス幅の値が小さいほど DAG に近い構造をしていることを表す．無向パス幅を求める問題が NP 困難であることより，一般の有向グラフに対して有向パス幅を求める問題もまた NP 困難である．

以下では DAG パス分解・DAG パス幅の定義を示す．

Definition 7 (DAG パス分解). [12] $G = (V, E)$ を有向グラフとする．G の DAG パス分解 (DAG-PD) とは，以下の 3 つの条件をみたすような V の部分集合 X_i $(i = 1, 2, \ldots, s)$ の列 $X = (X_1, X_2, \ldots, X_s)$ である．

1. $X_1 \cup X_2 \cup \cdots \cup X_s = V$
2. 任意の有向枝 $(u, v) \in E$ に対し，以下のいずれかが成り立つ．
   - $u, v \in X_1$
   - ある i $(i \geq 2)$ があり，$u, v \in X_i$, $v \notin X_{i-1}$
3. 任意の i, j, k $(1 \leq i \leq j \leq k \leq s)$ について，$X_i \cap X_k \subseteq X_j$．すなわち任意の頂点 $v \in V$ について，v は X 上でただ一つの非空なパスを誘導する．

なお，[12] では 1 を以下のように定義していることに注意する．本研究ではアルゴリズムの構築のしやすさの観点から上記の定義を用いるものとする．

1. ある i $(i \geq 2)$ があり，$u, v \in X_i$, $u \notin X_{i-1}$

以下では各頂点集合 X_i をバッグと呼ぶ．

Definition 8 (DAG パス幅). 有向グラフ G の DAG パス分解 $X = (X_1, X_2, \ldots, X_s)$ に対し，$\max_i\{|X_i| - 1\}$ を X の幅という．G の DAG パス幅とは，G の全ての DAG パス分解を考えたときの幅の最小値である．

DAG パス幅は有向グラフがどれだけ有向パスに近い構造をしているかを表す

パラメータであり，DAG パス幅の値が小さいほど有向パスに近い構造をしていることを表す．DAG パス分解のルール 3 より，任意の頂点 v に対して v を含むバッグは連結となる．これとルール 2 とより，枝 (u, v) に対して u, v はあるバッグ X_i に初めて同時に現れるか，u を含み v を含まないバッグが先に現れた後，u, v を同時に含むバッグが現れることを示している．すなわち DAG パス分解はグラフのトポロジカル順序でバッグに頂点を追加していく操作であることを示している．一般の有向グラフに対して DAG パス幅を求める問題は NP 困難であるが，DAG パス幅が 1 であるグラフクラスは以下のようなキャタピラ型の有向グラフであることを以下で示す．

Definition 9 (キャタピラ型)**.** 有向グラフ G がキャタピラ型であるとは，有向木であり，かつ G の頂点のうち入次数 1，出次数 0 の頂点を除くと単一のパスになるようなグラフである．

Lemma 1. 頂点数 n (> 2) の連結な有向グラフ G の *DAG* パス幅が *1* であることの必要十分条件は，G がキャタピラ型であることである．

Proof. G に入次数 2 以上の頂点 v が存在する場合，DAG パス分解のルール 2 より，G の任意の DAG パス分解には v とその先行頂点をすべて含むバッグが必ず存在するため，G の DAG パス幅は少なくとも 2 以上となる．したがって以下では G に入次数 2 以上の頂点が存在しない，すなわち有向木である場合のみを考える．まず G がキャタピラ型でないとき，G のパス幅が 2 以上となることを示す．そのために G の頂点数で場合分けをして考える．$|V[G]| = 2$ の場合，G は明らかにキャタピラ型である．$|V[G]| = 3, 4$ の場合もまた G はキャタピラ型である．以下に理由を示す．G が有向木ならば，G に含まれる最長のパスを 1 つ選択し，その頂点列を P とする．$|P| = 3$ の場合，G の枝分かれの数は高々1 つであり，その枝分かれの長さも高々1 である．したがって G はキャタピラ型である．$|P| = 4$ の場合，$|V[G]| = 4$ に注意すると G は長さ 4 のパスであるため，明らかにキャタピラ型である．以上より $|V[G]| = 3, 4$ の場合，G はキャタピラ型である．次に G の頂点数が 5 以上の場合を考える．ここで G がキャタピラ型でないとき，定義より G から出次数 0 の頂点を除いても枝分かれをもつ有向木を内部に含む．すなわちある頂点 $v \in V[G]$ が存在し，v を始点とする長さ 2 以上の点素なパスが少なくとも 2 つ存在する．このパスを $P_1 = v, v_1, v_2, \ldots$, $P_2 = v, u_1, u_2, \ldots$ とする．DAG パス分解のルールに注意すると，G の任意の

DAG パス分解には次の 3 つの頂点集合 $\{v, v_1, v_2\}, \{v, u_1, u_2\}, \{v, v_1, u_1\}$ のうち，少なくとも 1 つを含むバッグが必ず存在するため，DAG パス幅は 2 以上となる．したがって G の DAG パス幅が 1 であるためには G がキャタピラ型であることが必要．逆に G がキャタピラ型であるとき，G の各頂点 v は出次数が 0 の葉である子 $v_1, v_2, \ldots, v_s$ をもち，出次数が 1 以上の葉でない子を高々1 つもつ．v が葉でない子をもたないとき，順にバッグ $\{v, v_1\}, \{v, v_2\}, \ldots, \{v, v_s\}$ を生成することで v と $\mathsf{suc}(v)$ に対する DAG パス分解を構築できる．v が葉でない子 u をもつとき，順にバッグ $\{v, v_1\}, \{v, v_2\}, \ldots, \{v, v_s\}, \{v, u\}, \{u\}$ を生成することで，v と $\mathsf{suc}(v)$ に対する DAG パス分解を構築できる．これを全ての頂点に対して考えることにより，幅が 1 である G の DAG パス分解が得られる．したがって G がキャタピラ型であれば G の DAG パス幅は 1 である． □

以下では動的計画法を行いやすくするための DAG パス分解として nice DAG パス分解を定義する．

Definition 10 (nice DAG パス分解)**.** [12] DAG $G = (V, E)$ の DAG パス分解 $X = (X_1, X_2, \ldots, X_s)$ が nice DAG パス分解 (nice DAG-PD) であるとは，X が以下のルールを満たすことをいう．

1. $X_1 = X_s = \emptyset$
2. 任意の i $(2 \leq i \leq s-1)$ に対して，以下のいずれかが成り立つ．
   - (introduce) ある強連結成分 $S \subseteq V$ があり，$S \cap X_i = \emptyset$, $X_{i+1} = X_i \cup S$
   - (forget) ある頂点 $v \in V$ があり，$X_{i+1} = X_i \backslash \{v\}$

introduce は，あるバッグに強連結な頂点集合を追加したものを次のバッグとする操作であり，forget は，あるバッグから頂点を一つ取り除いたものを次のバッグとする操作である．G が DAG であるとき，強連結成分 S は 1 つの頂点のみからなるため，introduce の定義は以下のようになる．

1. (introduce) ある頂点 $v \in V$ があり，$\{v\} \cap X_i = \emptyset$, $X_{i+1} = X_i \cup \{v\}$

nice DAG パス分解は，各バッグが 1 つの頂点の introduce か forget かに限られるため，動的計画法の設計が容易になる利点がある．また [20] では，ある DAG パス分解 X に対し，X と同じ幅をもつ nice DAG-PD を多項式時間で構築できることを示しているほか，バッグの個数について以下を示している．

Proposition 1. 有向グラフ G に対する任意の DAG パス分解を $X = (X_1, X_2, \ldots, X_s)$ とする．各 i に対し $X_i \neq X_{i+1}$ ならば，$s \leq 2|V[G]| + 1$ が成り立つ．

以降ではバッグ X_{i+1} がある強連結成分 S を introduce しているとき，バッグ X_{i+1} は introduce である，という．同様にバッグ X_{i+1} がある頂点 v を forget しているとき，バッグ X_{i+1} は forget である，という．

2.3 Black Pebbling game

以下では DAG パス分解との比較を行うため，Black Pebbling game の定義を行う．

Definition 11 (Black Pebbling game)**.** [13] Black Pebbling game とは，DAG $G = (V, E)$ が与えられたときに，以下のルールを満たす戦略 $P = (P_1, P_2, \ldots, P_t)$ $(P_i \subseteq V)$ を構成するゲームである．

DAG G は出次数が 0 の頂点 z をただ一つもつとする．グラフ小石ゲームの戦略とは，以下の 4 つのルールを満たすような V の部分集合 P_i $(i = 0, 1, \ldots, s)$ の列 $P = (P_0, P_1, \ldots, P_\tau)$ である．

1. $P_0 = \emptyset, P_\tau = \{z\}$
2. pebble: $v \in V$ に小石が置かれておらず，かつ v の全ての先行頂点に小石が置かれていれば，v に小石を置いてもよい．すなわち $v \notin P_{i-1}$ かつ，任意の $(u, v) \in E$ に対し $u \in P_{i-1}$ ならば，$P_i = P_{i-1} \cup \{v\}$ とできる．
3. unpebble: v に置かれた小石はいつでも取り除いてもよい．すなわち $v \in P_{i-1}$ ならば，$P_i = P_{i-1} \backslash \{v\}$ とできる．
4. 一度小石を取り除いた頂点には，再び小石を置くことはできない．

pebble は頂点に小石を置く操作を表し，forget は頂点から小石を取り除く操作を表す．また，ペブリング数を以下のように定義する．

Definition 12 (ペブリング数)**.** DAG G の戦略 $P = (P_0, P_1, \ldots, P_\tau)$ に対し，*space* と *time* を以下のように定義する．

1. $\mathsf{space}(P) = \max_i \{|P_i|\}$
2. $\mathsf{time}(P) = \tau$

G のペブリング数とは，G の全ての戦略を考えたときの *space* の最小値である．

一般の DAG に対してペブリング数を求める問題は PSPACE 完全である [14]．以下では DAG G に対するペブリング数を $Peb(G)$ と表現する．グラフ小石ゲームは，Proof of Space [15] とよばれるブロックチェーン技術の中で用いられる．Proof of Space は空きディスク容量をいくら保持しているかを証明する手法であり，入力 DAG が空きディスク容量に対応する．またペブリング数は同時に使

用するメモリの量に対応する．ペブリング数が大きいほど，より多くのメモリやデータを使うため，その証明が難しいことを表す．すなわち証明の安全性が高くなることを表す．

また one-shot Black Pebbling (one-shot BP) とは，Black Pebbling game に対して以下のルールを追加したものである．

- DAG G の任意の頂点はちょうど 1 度だけ小石が置かれる．

one-shot BP についても同様にペブリング数を定めることができる．一般の DAG に対して one-shot BP のペブリング数を求める問題は NP 困難である [16]．4.3 節では one-shot BP が DAG 上の DAG パス分解を構成する問題と等価であることを示す．

第3章　DAGパス幅を用いた様々なNP困難問題に対するアルゴリズム

本節では，DAG 上の 4 つの NP 困難問題である Directed Dominating Set Problem，Max Leaf Outbranching Problem，Disjoint Path Problem，Directed Steiner Tree Problem の DAG パス幅を用いたパラメータ化アルゴリズムを提案する．これらの問題は有向木幅や有向パス幅に対しては $W[1]$-hard であるが [17, 21]，DAG パス幅を用いることで容易にパラメータ化アルゴリズムを構築することができる．なお，本論文では入力グラフを DAG としているが，一般の有向グラフに対しても強連結成分分解を行い DAG に変換することで本アルゴリズムを適応することが可能である．

3.1　Directed Dominating Set Problem の $O(2^w wn)$ 時間アルゴリズム

以下では，頂点数 n の DAG G と幅が w である G の DAG-PD が与えられたときに，G の Directed Dominating Set Problem を $O(2^w wn)$ で計算するアルゴリズムを示す．

まず Directed Dominating Set に関する定義と定理を与える．

Definition 13 (Directed Dominating Set)**.** 有向グラフ $G = (V, E)$ に対し，$S \subseteq V$ が G の Directed Dominating Set (DiDS) であるとは，任意の $v \in V \backslash S$ に対し，ある $u \in S$ があり，$(u, v) \in E$ を満たすことである．minimum DiDS (以下 mDiDS) とは，全ての DiDS S のうち $|S|$ が最小のものである．

DiDS problem とは，G の mDiDS のサイズを求める問題である．Ganian ら [17] は以下を示した．

Proposition 2 (計算量クラス)**.** *DiDS problem* は G が *DAG* の場合でも *NP* 困難である．

本節の以降では DAG パス幅を用いた FPT アルゴリズムの構築を行う．

Theorem 1. *DAG* G に対し，幅が w である G の *nice DAG-PD* が与えられたとき，G の *DiDS problem* を $O(2^w wn)$ で解くアルゴリズムが存在する．

上記のアルゴリズムを構成するため，まず関数 DS を定義する．

Definition 14 (DS)**.** DAG $G = (V, E)$ の nice DAG-PD を $P = (X_1, X_2, \ldots, X_s)$ とする．ある i $(i = 1, 2, \ldots, s)$ に対し，頂点集合 $A_i, B_i \subseteq V$ が $A_i \cup B_i =$

$X_i, A_i \cap B_i = \varnothing$ を満たすとする．G_i を頂点集合 $X_1 \cup X_2 \cup \cdots \cup X_i$ によって誘導される G の部分グラフとする．関数 DS を以下のように定める．

$$\mathsf{DS}(i, A_i, B_i) = \min \left\{ |S_i| \;\middle|\; \begin{array}{l} S_i \subseteq X_1 \cup X_2 \cup \cdots \cup X_i, \\ S_i \text{は} G_i \text{の DiDS} \\ A_i \subseteq S_i, B_i \cap S_i = \varnothing \end{array} \right\} \tag{1}$$

DS は G_i の mDiDS のサイズを計算する関数である．
以下で DS の計算式を与える．各 X_i が introduce か forget かで場合分けをして計算する．

- X_i が $v \in V$ を introduce しているとき

$$\mathsf{DS}(i, A_i, B_i) = \begin{cases} \mathsf{DS}(i-1, A_i \backslash \{v\}, B_i) + 1 & (v \in A_i) \\ \mathsf{DS}(i-1, A_i, B_i \backslash \{v\}) & (v \in B_i \text{かつ } \mathsf{pred}(v) \cap A_i \neq \varnothing) \\ \infty & (otherwise) \end{cases}$$

- X_i が $v \in V$ を forget しているとき

$$\mathsf{DS}(i, A_i, B_i) = \min\{\mathsf{DS}(i-1, A_i \cup \{v\}, B_i), \mathsf{DS}(i-1, A_i, B_i \cup \{v\})\}$$

DS を用いて，DAG G の nice DAG-PD P が与えられたときに，G の mDiDS のサイズを出力するアルゴリズム Compute を示す．

Compute(P)

1. First Step: $\mathsf{DS}(0, \varnothing, \varnothing) = 0$ とする．
2. Exection Step: P の各 X_i $(i = 1, 2, ..., s)$ に対し，A_i, B_i 全ての組合せについて順に $\mathsf{DS}(i, A_i, B_i)$ を計算する．
3. Final Step: $i = s$ ならば，$\mathsf{DS}(s, \varnothing, \varnothing)$ を出力する．

Lemma 2. Compute は G の *mDiDS* のサイズを返す．

Proof. 各 i に対し，$\mathsf{DS}(i, A_i, B_i)$ が DS の定義 1を満たすことを示せば十分．これを i に関する数学的帰納法で示す．$i = 0$ のとき，明らかに定義 1を満たす．$i = k$ のとき，$\mathsf{DS}(i, A_i, B_i)$ が定義 1を満たすと仮定する．以下で X_{k+1} が $v \in V$ を introduce するか forget するかで場合分けを行う．

- X_{k+1} が $v \in V$ を introduce する場合
  $v \in A_{k+1}$ の場合，DAG-PD のルール 2 より v は G_{k+1} のどの頂点も支配しないことに注意すると，$\mathsf{DS}(k+1, A_{k+1}, B_{k+1})$ は $\mathsf{DS}(k, A_{k+1}\backslash\{v\}, B_{k+1}) + |v|$ と等しい．仮定より $\mathsf{DS}(k, A_{k+1}\backslash\{v\}, B_{k+1})$ は $A_{k+1}\backslash\{v\}$ を含み，B_{k+1} を含まないような，G_k の mDiDS のサイズと等しいから，$\mathsf{DS}(k+1, A_{k+1}, B_{k+1})$ は A_{k+1} を含み，B_{k+1} を含まないような，G_{k+1} の mDiDS のサイズと等しい．したがって定義 1が成立．
  $v \in B_{k+1}$ かつ $\mathsf{pred}(v) \cap A_{k+1} \neq \varnothing$ の場合，ある頂点 u $((u, v) \in G_{k+1}, u \in A_{k+1})$ が存在するため v は u によって支配されている．したがって $\mathsf{DS}(k+1, A_{k+1}, B_{k+1})$ は $\mathsf{DS}(k, A_{k+1}, B_{k+1}\backslash\{v\})$ と等しい．仮定より $\mathsf{DS}(k, A_{k+1}, B_{k+1}\backslash\{v\})$ は A_{k+1} を含み，B_{k+1} を含まないような，G_k の mDiDS のサイズと等しいから，$\mathsf{DS}(k+1, A_{k+1}, B_{k+1})$ は A_{k+1} を含み，B_{k+1} を含まないような，G_{k+1} の mDiDS のサイズと等しい．したがって定義 1が成立．
  $v \in B_{k+1}$ かつ $\mathsf{pred}(v) \cap A_{k+1} = \varnothing$ の場合，v を支配する頂点が A_{k+1} に存在しない．したがって定義 1を満たすような mDiDS が存在しないため，$\mathsf{DS}(k+1, A_{k+1}, B_{k+1})$ の値を ∞ とすることでこれを表している．
- X_{k+1} が $v \in V$ を forget する場合
  $G_{k+1} = G_k$ より，G_{k+1} の mDiDS は G_k の mDiDS と等しい．したがって v を含むような G_k の mDiDS のサイズと，v を含まないような G_k の mDiDS のサイズうち，値の小さい方を G_{k+1} の mDiDS のサイズとすることができる．仮定よりそれぞれ $\mathsf{DS}(k, A_{k+1} \cup \{v\}, B_{k+1}), \mathsf{DS}(k, A_{k+1}, B_{k+1} \cup \{v\})$ と表すことができるため，

$$\min\{\mathsf{DS}(k, A_{k+1} \cup \{v\}, B_{k+1}), \mathsf{DS}(k, A_{k+1}, B_{k+1} \cup \{v\})\}$$

  は G_{k+1} の mDiDS のサイズと等しい．したがって定義 1が成立．

以上より，$i = k+1$ でも定義 1が成立．数学的帰納法により Lemma 2が証明された． □

最後に Compute の計算量を示す．

Lemma 3. *DAG G の頂点数が n であるとする．このとき，幅が w である G の DAG-PD P が与えられたとき，$\mathsf{Compute}(P)$ は $O(2^w wn)$ の計算時間で結果を出力する．*

Proof. 各 X_i に対し，$|X_i| \leq w+1$ に注意すると，A, B の組合せは高々2^{w+1} 通り存在する．また Proposition 1 より $0 \leq i \leq 2|V|+1$ である．さらに $\mathsf{pred}(v) \cap A_i \neq \varnothing$ の判定に高々$O(w)$ 時間を要することに注意すると，$\mathsf{DS}(i, A_i, B_i)$ の計算量は，X_i が introduce ならば $O(w)$ であり，forget ならば $O(1)$ である．以上より，Compute の計算量は $O(2^w wn)$ である． □

3.2 Max Leaf Outbranching の $O(2^w wn + n^2)$ 時間アルゴリズム

本節では頂点数 n の DAG G, 幅が w である G の DAG-PD, 根 r が与えられたとき，G の Max Leaf Outbranching Problem を $O(2^w wn + n^2)$ で計算するアルゴリズムを示す．

以下で Max Leaf Outbranching Problem に関する定義と定理を与える．

Definition 15 (Max Leaf Outbranching Problem (MaxLOB))**.** 有向グラフ $G = (V, E)$, 根 $r \in V$ が入力されたとき，G の有向全域木のうち葉数が最大となる有向全域木 T の葉数を求める問題である．

Ganian ら [17] は以下を示した．

Proposition 3 (計算量クラス)**.** *MaxLOB* は G が *DAG* の場合でも *NP* 困難である．

今回は入力グラフが DAG である場合を考える．

本節の以降では DAG パス幅を用いた FPT アルゴリズムの構築を行う．

Theorem 2. *DAG* G に対し，幅が w である G の *nice DAG-PD* が与えられたとき，G の *MaxLOB* を $O(2^w wn + n^2)$ で解くアルゴリズムが存在する．

上記のアルゴリズムを構成するため，まず関数 LOB を定義する．

Definition 16 (LOB)**.** DAG $G = (V, E)$ の nice DAG-PD を $P = (X_1, X_2, \ldots, X_s)$ とする．ある i $(i = 1, 2, \ldots, s)$ に対し，頂点集合 $A_i, B_i \subseteq V$ が $A_i \cup B_i = X_i, A_i \cap B_i = \varnothing$ を満たすとする．G_i を頂点集合 $X_1 \cup X_2 \cup, \cdots \cup X_i$ によって誘導される G の部分グラフとする．関数 LOB を以下のように定める．ただし有向木 T の葉の集合を $\mathsf{Leaf}(T)$ と表す．

$$\mathsf{LOB}(i, A_i, B_i) = \max \left\{ |\mathsf{Leaf}(T_i)| \;\middle|\; \begin{array}{l} T_i = (V[T_i], E[T_i]) \text{ は } r \text{ を根とする } G_i \text{の有向全域木} \\ V[T_i] = V[G_i], E[T_i] \subseteq E[G_i] \\ A_i \subseteq \mathsf{Leaf}(T_i), B_i \subseteq V[T_i] \backslash \mathsf{Leaf}(T_i) \end{array} \right\} \tag{2}$$

LOB は G_i での MaxLOB を計算する関数である．以下で LOB の計算式を与える．各 X_i が introduce か forget かで場合分けをして計算する．

- X_i が $v \in V$ を introduce しているとき

$$\mathsf{LOB}(i, A_i, B_i) = \begin{cases} \mathsf{LOB}(i-1, A_i \backslash \{v\}, B_i) + 1 & (v \in A_i \text{かつ } \mathsf{pred}(v) \cap B_i \neq \varnothing) \\ \mathsf{LOB}(i-1, A_i, B_i \backslash \{v\}) & (v \in B_i \text{かつ } \mathsf{pred}(v) \cap B_i \neq \varnothing) \\ -\infty & (otherwise) \end{cases} \tag{3}$$

- X_i が $v \in V$ を forget しているとき

$$\mathsf{LOB}(i, A_i, B_i) = \max\{\mathsf{LOB}(i-1, A_i \cup \{v\}, B_i), \mathsf{LOB}(i-1, A_i, B_i \cup \{v\})\} \tag{4}$$

LOB を用いて，DAG G の nice DAG-PD P が与えられたときに，G の MaxLOB の解を出力するアルゴリズム Compute を示す．

Compute(P)

1. Preprocessing: $G = (V, E)$ において，r から到達可能な頂点集合を V_r とする．P の各バッグ X_i に対し，任意の $v \in V \backslash V_r$ を X_i から取り除く．こうしてできる頂点集合の列を nice DAG-PD に変換したものを，便宜上新たに $P = (X_1, X_2, \ldots, X_s)$ とする．
2. First Step: $V_r = \{r\}$ ならば 1 を解として出力する．$V_r \neq \{r\}$ ならば $\mathsf{LOB}(1, \{r\}, \varnothing) = -\infty, \mathsf{LOB}(1, \varnothing, \{r\}) = 0$ とする．
3. Exection Step: P の各 X_i $(i = 1, 2, ..., s)$ に対し，A_i, B_i 全ての組合せについて順に $\mathsf{LOB}(i, A_i, B_i)$ を計算する．
4. Final Step: $i = s$ ならば，$\mathsf{LOB}(s, \varnothing, \varnothing)$ を出力する．

Lemma 4. Compute は G の *MaxLOB* の解を出力する．

4の証明は付録で示す.

最後に Compute の計算量を示す.

Lemma 5. *DAG G の頂点数が n であるとする．このとき，幅が w である G の DAG-PD P が与えられたとき，*Compute(P) *は $O(2^w wn + n^2)$ の計算時間で結果を出力する.*

Proof. Preprocessing において，r から到達可能な頂点集合の計算に $O(n^2)$ かかる．First Step, Final Step は各々O(1) で計算できる．以下で Exection Step での計算量を考える．各 X_i に対し，$|X_i| \leq w+1$ に注意すると，A, B の組合せは高々2^{w+1} 通り存在する．また Proposition 1より $0 \leq i \leq 2|V|+1$ である．さらに pred(v) の計算量が高々$O(w)$ であることに注意すると，LOB(i, A_i, B_i) の計算量は，X_i が introduce ならば $O(w)$ であり，forget ならば $O(1)$ である．以上より，Compute の計算量は $O(w2^w n + n^2)$ である． □

3.3 Disjoint Path の $O((k+1)^w(w^2+k)n + n^2)$ 時間アルゴリズム

本節では頂点数 n の DAG G, 幅が w である G の DAG-PD, k 個の頂点対が与えられたとき，G の Disjoint Path Problem を $O((k+1)^w(w^2+k)n + n^2)$ で計算するアルゴリズムを示す.

以下で Disjoint Path Problem に関する定義と定理を与える.

Definition 17 (Disjoint Path Problem)**.** DAG $G = (V, E)$, k 個の頂点対 $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ が入力されたとき，各 s_i から t_i までの点素なパスの組を $\mathcal{P} = (P_1, P_2, \ldots, P_k)$ とする．各パス P_i の長さを P_i の頂点数 $|P_i|$ としたとき，Disjoint Path Problem はパスの合計長 $\sum_{i=1}^{k} |P_i|$ の最小値を求める問題である.

Ganian ら [17] は以下を示した.

Proposition 4 (計算量クラス)**.** *Disjoint Path problem* は *NP* 困難である.

本節の以降では DAG パス幅を用いた XP アルゴリズムの構築を行う.

Theorem 3. *DAG G に対し，幅が w である G の nice DAG-PD が与えられたとき，G の Disjoint Path Problem を $O((k+1)^w(w^2+wk)n + n^2)$ で解くアルゴリズムが存在する.*

上記のアルゴリズムを構成するため，まず関数 Cal を定義する.

Definition 18 (Cal). DAG $G = (V, E)$ の nice DAG-PD を $X = (X_1, X_2, \ldots, X_s)$ とする．ある i $(i = 1, 2, \ldots, s)$ に対し，頂点集合 $A_i^1, A_i^2, \ldots, A_i^k, B_i \subseteq V$ が $A_i^1 \cup A_i^2 \cup \cdots \cup A_i^k \cup B_i = X_i$ を満たし，さらに $A_i^1, A_i^2, \ldots, A_i^k, B_i$ のうち任意の 2 つの頂点集合は共通部分集合を持たないとする．G_i を頂点集合 $X_1 \cup X_2 \cup, \cdots \cup X_i$ によって誘導される G の部分グラフとする．$\mathscr{A}_i = (A_i^1, A_i^2, \ldots, A_i^k)$ とし，関数 Cal を以下のように定める．

$$\mathsf{Cal}(i, \mathscr{A}_i, B_i) = \min \sum_{m=1}^{k} (|P_i^m| - 1) \tag{5}$$

ただし，頂点集合 $P_i^m (i \leq m \leq k)$ は $P_i^m \subseteq X_1 \cup X_2 \cup, \cdots \cup X_i$ を満たし，それぞれ s_m を始点とする点素なパスを構成し，$m' \neq m$ として $A_i^m \subseteq P_i^m, A_i^{m'} \cap P_i^m = \varnothing, B_i \cap P_i^m = \varnothing$ を満たす．このとき Cal は G_i の各 s_m を始点とする k 個の点素なパスの合計長の最小値を計算する関数である．

以下で Cal の計算式を与える．各 X_i が introduce か forget かで場合分けをして計算する．ただし頂点対の始点と終点の集合をそれぞれ $S = \{s_1, s_2, \ldots, s_k\}, T = \{t_1, t_2, \ldots, t_k\}$ とする．

- X_i が $v \in S$ を introduce しているとき $(v = s_m$ とする$)$

$$\mathsf{Cal}(i, \mathscr{A}_i, B_i) = \begin{cases} 0 & (A_i^m = \{v\}) \\ \infty & (otherwise) \end{cases} \tag{6}$$

- X_i が $v \in T$ を introduce しているとき $(v = t_m$ とする$)$

$$\mathsf{Cal}(i, \mathscr{A}_i, B_i) = \begin{cases} \mathsf{Cal}(i-1, \mathscr{A}_i^m, B_i) + 1 & (v \in A_i^m \text{かつ，ある } w \in \mathsf{pred}(v) \cap A_i^m \\ & \text{が存在し,} \mathsf{suc}(w) \cap A_i^m = \{v\}) \\ \infty & (otherwise) \end{cases} \tag{7}$$

• X_i が $v \in V \backslash (S \cup T)$ を introduce しているとき

$$\mathsf{Cal}(i, \mathscr{A}_i, B_i) = \begin{cases} \mathsf{Cal}(i-1, \mathscr{A}_i^m, B_i) + 1 & (v \in A_i^m \text{かつ，ある } w \in \mathsf{pred}(v) \cap A_i^m \\ & \text{が存在し,} \mathsf{suc}(w) \cap A_i^m = \{v\}) \\ \mathsf{Cal}(i-1, \mathscr{A}_i^m, B_i \backslash \{v\}) & (v \in B_i) \\ \infty & (otherwise) \end{cases} \tag{8}$$

• X_i が $v \in V$ を forget しているとき

$$\mathsf{Cal}(i, \mathscr{A}_i, B_i) = \min\{\min_{1 \le m \le k} \{\mathsf{Cal}(i-1, \overline{\mathscr{A}}_i^m, B_i)\}, \mathsf{Cal}(i-1, \mathscr{A}_i^m, B_i \cup \{v\})\} \tag{9}$$

ただし，$v \in V$ に対し $\mathscr{A}_i^m, \overline{\mathscr{A}}_i^m$ を以下のように定める．

$$\mathscr{A}_i^m = (A_i^1, A_i^2, \ldots, A_i^m \backslash \{v\}, \ldots, A_i^k)$$
$$\overline{\mathscr{A}}_i^m = (A_i^1, A_i^2, \ldots, A_i^m \cup \{v\}, \ldots, A_i^k)$$

Cal を用いて，DAG G の nice DAG-PD P が与えられたときに，G の Disjoint Path Problem の解を出力するアルゴリズム Compute を示す．

Compute(P)

1. Preprocessing: 入力グラフが単一頂点 $s_1 = t_1$ からなる場合，0 を出力する．そうでない場合，各頂点 $t \in T$ に入る枝をすべて削除する．こうしてできるグラフを便宜上新たに G とする．
2. First Step: $\mathsf{Cal}(0, (\varnothing, \varnothing, \ldots, \varnothing), \varnothing) = 0$ とする．
3. Exection Step: P の各 X_i $(i = 1, 2, \ldots, s)$ に対し，$\mathscr{A}_i, B_i$ 全ての組合せについて順に $\mathsf{Cal}(i, \mathscr{A}_i, B_i)$ を計算する．
4. Final Step: $i = s$ ならば，$\mathsf{Cal}(s, (\varnothing, \varnothing, \ldots, \varnothing), \varnothing)$ を出力する．

Lemma 6. Compute は G の *Disjoint Path Problem* の解を出力する．

Proof. 入力グラフ G が単一頂点 $s_1 = t_1$ からなる場合，Compute の Preprocessing の処理により 0 を出力する．これは明らかに G の Disjoint Path になっている．以下では G が単一頂点から成らない場合を考える．あるバッグ X_i が $v \in V$ を introduce するとき，v は $A_i^1, A_i^2, \ldots, A_i^k, B_i$ のいずれかにのみ含まれるため，これを 1 から i まで考えることにより各パス P_i^m は互いに共通の頂点を持たない点素な

パスであることが示される．また，ある始点 $s_m \in S$ が introduce されたバッグを X_{i_s} とすると，$X_{i'}$ $(i' < i_s)$ で introduce された頂点はパス $P^m_{i_s}$ に含まれることはない．なぜならば，$X_{i'}$ で頂点 u が introduce され，かつ $A^m_{i'} = \{u\}$ であるとすると，$\mathsf{pred}(u) \cap A^m_{i'} = \varnothing$ であり，式8の条件より Cal の値は ∞ となるからである．さらに，ある終点 $t_m \in S$ が introduce されたバッグを X_{i_t} とすると，$X_{i'}$ $(i_t < i')$ で introduce される頂点はパス $P^m_{i'}$ に含まれることはない．なぜならば，X_j $(i_t < j)$ で頂点 w が introduce され，かつ Compute の Preprocessing を行う前の入力グラフにおいて $w \in \mathsf{suc}(t_m)$ であるとする．ここで $w \in A^m_j$ であるとすると，DAG-PD のルール 2 より $t_m \in A^m_j$ がいえるが，もし $\mathsf{pred}(w) \cap (A^m_j \backslash \{t_m\}) = \varnothing$ ならば，Preprocessing の処理により $\mathsf{pred}(w) \cap A^m_j = \varnothing$ であるため，式8の条件より Cal の値は ∞ となる．一方，$\mathsf{pred}(w) \cap (A^m_j \backslash \{t_m\}) \neq \varnothing$ ならば，ある頂点 $p \in \mathsf{pred}(w) \cap A^m_j$ $(p \neq t_m)$ が存在し，式8の条件より，p のある後続頂点 $q \in A^m_j$ $(p \in \mathsf{pred}(q) \cap (A^m_j \backslash \{w\})$ かつ $\mathsf{suc}(p) \cap (A^m_j \backslash \{w\}) = \{q\})$ が存在する．Preprocessing の処理によって枝 (t_m, w) が除かれることに注意すると $\{q, w\} \subseteq \mathsf{suc}(p) \cap A^m_j$ が成り立つため，w の introduce において式8の条件より Cal の値は ∞ となる．これ以降のバッグ X_l $(j < l)$ で introduce された頂点 w' が A^m_l に含まれた場合，枝 (w, w') が存在すれば式8の条件より Cal の値は ∞ となり，枝 (w, w') が存在しなければ上記と同様の議論により Cal の値は ∞ となる．以上より X_{i_t} 以降で introduce された頂点はパス $P^m_{i_t}$ に追加されることはない．

以上の議論に注意すると，Lemma 6の証明では，各 i $(i_s \leq i \leq i_t)$ に対し，ある m $(1 \leq m \leq k)$ について $\mathsf{Cal}(i, \mathscr{A}_i, B_i)$ が Cal の定義5を満たすことを示せば十分．これを i に関する数学的帰納法で示す．

$i = i_s$ のとき，式6より $A^m_{i_s} = \{s_m\}$ の場合は 0 であり，これは s_m のみからなるパス $P^m_{i_s}$ の長さが 0 であることを表す．これは明らかに定義5を満たす．$A^m_{i_s} \neq \{s_m\}$ の場合は ∞ であるが，これは s_m を始点とするパスが構成されないことを表すため，同様に定義5を満たす．

ある i $(i_s \leq i < i_t)$ において，$\mathsf{Cal}(i, \mathscr{A}_i, B_i)$ が定義5を満たすと仮定する．以下で X_{i+1} が $v \in V$ を introduce するか forget するかで場合分けを行う．

- X_{i+1} が $v \in V$ を introduce する場合
  まず $v \in A^m_{i+1}$ の場合を考える．A^m_{i+1} に含まれ，かつ v の直前に introduce された頂点を u とする．DAG-PD のルール 2 より，u, v の間には枝 (u, v) が存在するか，(u, v) は存在せず u 以外の頂点 u' $(u' \in A^m_{i+1})$ との間の枝 (u', v)

が存在するか，A^m_{i+1} に含まれるどの頂点との間にも枝が存在しないかのいずれかである．枝 (u, v) が存在する場合，式 8より $\mathsf{Cal}(i+1, \mathscr{A}_{i+1}, B_{i+1}) = \mathsf{Cal}(i, \mathscr{A}^m_{i+1}, B_{i+1}) + 1$ となる．DAG-PD のルール 2 より u はパスの端点であることに注意すると，仮定より $\mathsf{Cal}(i, \mathscr{A}^m_{i+1}, B_{i+1})$ は m 番目のパスが s_m から u までのパスであるときの全パスの合計長の最小値を表している．したがって m 番目のパスの端点 u に v を加えることで合計長を 1 だけ大きくしており，$\mathsf{Cal}(i+1, \mathscr{A}_{i+1}, B_{i+1})$ は明らかに全パスの合計長の最小値を表す．よって定義 5を満たす．枝 (u, v) が存在せず u 以外の頂点 u' $(u' \in A^m_{i+1})$ との間の枝 (u', v) が存在する場合，上記の t_m と同様の議論により，ある頂点 $v' \in A^m_{i+1}$ $(v' \neq v)$ が存在し，$\{v, v'\} \subseteq \mathsf{suc}(u') \cap A^m_{i+1}$ が成り立つ．したがって式 8の条件より Cal の値は ∞ となる．これは A^m_{i+1} において枝分かれ $\{(u', v),\ (u', v')\}$ が存在し，これらを含むようなパスが存在しないことを表す．したがって定義 5を満たす．v と A^m_{i+1} に含まれるどの頂点との間にも枝が存在しない場合，式 8の条件より Cal の値は ∞ となる．これは A^m_{i+1} に含まれる頂点を使って s_m と v をつなぐパスを構成することができないことを表す．したがって定義 5を満たす．

$v \in B_{i+1}$ の場合，$v \notin A^m_{i+1}$ より，P^m_{i+1} は s_m から u までのパスであり，仮定よりこのときのパスの合計長の最小値は $\mathsf{Cal}(i, \mathscr{A}^m_{i+1}, B_{i+1} \backslash \{v\})$ で表される．したがって式 8より $\mathsf{Cal}(i+1, \mathscr{A}^m_{i+1}, B_{i+1})$ は v がどのパスにも含まれないときのパスの合計長の最小値を表すため，定義 5を満たす．

- X_{i+1} が $v \in V$ を forget する場合

  $G_{i+1} = G_i$ より，G_{i+1} と G_i でのパスの合計長の最小値は等しい．したがって v が $P^1_i, \ldots, P^k_i$ のいずれかのパスに含まれていた場合と，いずれにも含まれていない，すなわち B_i に含まれている場合とを考え，より値の小さいものを $i+1$ における Cal の値とすることができる．仮定よりそれぞれ $\mathsf{Cal}(i, \overline{\mathscr{A}}^m_{i+1}, B_{i+1}), \mathsf{Cal}(i, \mathscr{A}^m_{i+1}, B_{i+1} \cup \{v\})$ と表すことができるため，式 9は定義 5を表すようなパスの合計長の最小値を出力する．

以上より，$i = i+1$ でも定義 5が成立．数学的帰納法により Lemma 6が証明された． □

最後に Compute の計算量を示す．

Lemma 7. *DAG G の頂点数が n であるとする．このとき，幅が w である G の DAG-PD P が与えられたとき，*Compute(P) *は $O((k+1)^w(w^2+k)n+n^2)$ の計算時間で結果を出力する．*

Proof. Preprocessing において，G が単一頂点からなるかどうかは $O(1)$ で判定できる．また各 $t \in T$ に入るすべての枝の削除は $O(n^2)$ かかる．First Step, Final Step は各々O(1) で計算できる．以下で Exection Step での計算量を考える．各 X_i に対し，$|X_i| \leq w+1$ に注意すると，$A_i^1, A_i^2, \ldots, A_i^k, B_i$ の組合せは高々$(k+1)^{w+1}$ 通り存在する．また Proposition 1より $0 \leq i \leq 2|V|+1$ である．さらに pred(v), suc(v) の計算量がそれぞれ高々$O(w)$ であることに注意すると，Cal$(i, \mathscr{A}_i, B_i)$ の計算量は，X_i が introduce ならば $O(w^2)$ であり，forget ならば $O(k)$ である．以上より，Compute の計算量は $O((k+1)^w(w^2+k)n+n^2)$ である． □

また上記のアルゴリズムに変更を加えることで，辺素パス問題や誘導点素パス問題を解くアルゴリズムを構築することができる．

3.4 有向シュタイナー木問題の $O(2^w(k+w)n+n^2)$ 時間アルゴリズム

本節では，頂点数 n の DAG G と幅が w である G の DAG-PD，頂点集合 $R = \{t_1, t_2, \ldots, t_k\}$ が与えられたとき，R を含むような有向シュタイナー木問題 (Directed Steiner Tree Problem) を $O(2^w(k+w)n+n^2)$ で解く FPT アルゴリズムを示す．

以下で Directed Steiner Tree に関する定義と定理を与える．

Definition 19 (Directed Steiner Tree)**.** 枝重み付き有向グラフ $G=(V,E)$，根 $r \in V$，ターミナル $R = \{t_1, t_2, \ldots, t_k\} \subseteq V$ が与えられたとき，r を根とし，各 t_i を含む有向木を Directed Steiner Tree (DST) という．minimum-DST とは，すべての DST のうち枝の総重み (単に総重みと呼ぶ) が最小のものである．

DST problem とは，G の minimum-DST の総重みを求める問題である．各枝の重みが 1 である場合の DST problem を unit-cost DST problem と呼ぶ．Ganian ら [17] は unit-cost DST problem が DAG の場合でも NP 困難であることを示した．これより，ただちに DAG 上の DST problem が NP 困難であることが示される．

Fomin ら [18] は枝重み付き無向シュタイナー木問題に対し，ターミナル R のサイズ k をパラメータとした $2^{O(k \log k)}$ の FPT アルゴリズムを与えた．これは DST problem に拡張可能である．一方で，入力グラフの基礎グラフに対する木幅をパラメータとした FPT アルゴリズムは，我々の知る限り示されていない．本研究では DAG パス幅 w をパラメータとする FPT を構築し，計算量が $O(2^w(k+w)n+n^2)$ であることを示す．

Theorem 4. *DAG G に対し，ターミナルのサイズ $k=|R|$，幅が w である G の nice DAG-PD が与えられたとき，G の DST problem を $O(2^w(k+w)n+n^2)$ で解く FPT アルゴリズムが存在する．*

上記のアルゴリズムを構成するため，まず関数 ST を定義する．ただし $d(e)$ を枝 e の重みとする．

Definition 20 (ST)**.** DAG $G=(V,E)$ の nice DAG-PD を $P=(X_1,X_2,\ldots,X_s)$ とする．ある i $(i=1,2,\ldots,s)$ に対し，頂点集合 $A_i,B_i \subseteq V$ が $A_i \cup B_i = X_i, A_i \cap B_i = \varnothing$ を満たすとする．G_i を頂点集合 $X_1 \cup X_2 \cup, \cdots \cup X_i$ によって誘導される G の部分グラフとする．関数 ST を以下のように定める．

$$\mathsf{ST}(i;A_i,B_i) = \min\left\{ \sum_{(u,v)\in E[G_{T_i}]} d(u,v) \;\middle|\; \begin{array}{l} T_i \subseteq X_1 \cup X_2 \cup, \cdots \cup X_i \\ G_{T_i} \text{は} r \text{を根とする} G_i \text{上の有向木} \\ V[G_{T_i}]=T_i, E[G_{T_i}] \subseteq E[G_i] \\ A_i \subseteq T_i, B_i \cap T_i = \varnothing \\ {}^{\forall}t \in R \cap G_i \text{に対し}, t \in T_i \end{array} \right\} \tag{10}$$

各 i において，$\mathsf{ST}(i;A_i,B_i)$ の総重みをもち，上記の条件を満たす有向木 G_{T_i} が存在する場合，その有向木を $G^{opt}_{T(i,A_i,B_i)}$ と表す．このとき，$G^{opt}_{T(s,\varnothing,\varnothing)}$ は r を根とし，ターミナル R をすべて含む G の minimum-DST である．

以下で ST の計算式を与える．各 X_i が introduce か forget かで場合分けをして計算する．ただし入力されるグラフが DAG であるため，nice DAG-PD において introduce される強連結成分はただ一つの頂点からなることに注意する．

• X_i が $v \in V$ を introduce しているとき

$$\mathsf{ST}(i; A_i, B_i) = \begin{cases} \mathsf{ST}(i-1; A_i \backslash \{v\}, B_i) + \min\limits_{w \in \mathsf{pred}(v) \cap A_i} d(w, v) & (v \in A_i \text{かつ } \mathsf{pred}(v) \cap A_i \neq \varnothing) \\ \mathsf{ST}(i-1; A_i, B_i \backslash \{v\}) & (v \in B_i \text{かつ } v \notin R \cup \{r\}) \\ \infty & (otherwise) \end{cases} \tag{11}$$

• X_i が $v \in V$ を forget しているとき

$$\mathsf{ST}(i; A_i, B_i) = \min\{\mathsf{ST}(i-1; A_i \cup \{v\}, B_i), \mathsf{ST}(i-1; A_i, B_i \cup \{v\})\} \tag{12}$$

DAG G の nice DAG-PD P，根 r，ターミナル R が与えられたとき，ST を用いて R を含む G の minimum-DST の総重みを出力するアルゴリズム Compute を示す．

Compute(P, r, R)

1. Preprocessing: G において，r から到達可能な頂点集合を V_r とする．P の各バッグ X_i に対し，任意の $v \in V \backslash V_r$ を X_i から取り除く．こうしてできる頂点集合の列を nice DAG-PD に変換したものを，便宜上新たに $P' = (X_1, X_2, \ldots, X_{s'})$ とする．
2. First Step: $\mathsf{ST}(1; \{r\}, \varnothing) = 0$ とする．
3. Exection Step: P' の各 X_i $(i = 2, 3, \ldots, s')$ に対し，A_i, B_i 全ての組み合わせについて順に $\mathsf{ST}(i; A_i, B_i)$ を計算する．
4. Final Step: $i = s'$ ならば，$\mathsf{ST}(s'; \varnothing, \varnothing)$ を出力する．

Lemma 8. Compute は r を根とし，R をすべて含むような G の *minimum-DST* の総重みを出力する

Proof. まず，Preprocessing を行っても解が変わらないことを示す．Preprocessing における V_r は根 r から到達可能な頂点の集合であるため，任意の $v \in V \backslash V_r$ は目的の minimum-DST に含まれない．したがって考えるグラフを $G' = G[V \backslash V_r]$ としても解は変わらず，また P' は G' の nice DAG-PD となっていることに注意すると，ST に入力する nice DAG-PD を P' としても解は変わらない．また P' の幅は P の幅より大きくならないことに注意する．

次に，Preprocessing 後の処理で Compute が R をすべて含むような G の minimum-DST を出力することを示す．これは各 i について，式 (11)(12) が式 (10) を表し

ていることを示せば十分．これを $i \geq 1$ に関する数学的帰納法で示す．

$i = 1$ のとき，$\mathsf{ST}(1; \{r\}, \varnothing) = 0$ は明らかに式 (10) を表す．

$i = k$ $(1 \leq k < s')$ のとき，式 (11)(12) が式 (10) を表していると仮定する．以下で $i = k + 1$ についても式 (11)(12) が式 (10) を表すことを示す．これを X_{k+1} が，ある $v \in V$ を introduce するか forget するかで場合分けして考える．

- X_{k+1} が $v \in V$ を introduce する場合

  $v \in A_{k+1}, v \in B_{k+1}$ に場合分けして考える．$v \in A_{k+1}$ の場合，$\mathsf{pred}(v) \cap A_{k+1} = \varnothing$ ならば，A_i において v の先行頂点は 1 つも存在しない．このとき，DAG-PD のルール 2 より v の先行頂点は X_{k+1} 以降で introduce されないため，v を含む有向木は v が根の一つになっている．これは r をただ一つの根とする有向木になり得ないため，$\mathsf{ST}(k+1; A_{k+1}, B_{k+1}) = \infty$ とすることで式 (10) で表される有向木 $G^{opt}_{T(k,A_{k+1},B_{k+1})}$ が存在しないことを示している．一方，$\mathsf{pred}(v) \cap A_{k+1} \neq \varnothing$ ならば，v の先行頂点 $w \in A_{k+1} \backslash \{v\}$ が少なくとも 1 つ存在する．$i = k$ において式 (10) のような有向木 $G^{opt}_{T(k,A_{k+1}\backslash\{v\},B_{k+1})}$ が存在する場合，$w \in G^{opt}_{T(k,A_{k+1}\backslash\{v\},B_{k+1})}$ であるが，Preprocessing の操作より w は r から到達可能である．したがって v もまた r から w を通過して到達可能であり，$G^{opt}_{T(k,A_{k+1}\backslash\{v\},B_{k+1})\cup\{v\}}$ は r を根とする有向木である．またそのときの最小の総重みは $\sum_{(u,v)\in E[G^{opt}_{T(k,A_{k+1}\backslash\{v\},B_{k+1})}]} d(u,v) + \min_{w\in\mathsf{pred}(v)\cap A_i} d(w,v)$ と等しい．仮定より $\mathsf{ST}(k; A_k, B_k) = \sum_{(u,v)\in E[G^{opt}_{T(k,A_k,B_k)}]} d(u,v)$ であり，また $A_{k+1}\backslash\{v\} = A_k, B_{k+1} = B_k$ に注意すると，$\mathsf{ST}(k+1; A_{k+1}, B_{k+1})$ は式 (10) を表す．

  次に $v \in B_{k+1}$ の場合を考える．$v \in R \cup \{r\}$ のとき $\mathsf{ST}(k+1; A_{k+1}, B_{k+1}) = \infty$ であるが，これは式 (10) を表すような有向木 $G^{opt}_{T(k,A_{k+1},B_{k+1})}$ が存在しないことを示している．$v \notin R \cup \{r\}$ ならば．式 (10) より v は解となる有向木に含まれない．このとき $G^{opt}_{T(k+1,A_{k+1},B_{k+1}\backslash\{v\})} = G^{opt}_{T(k,A_k,B_k)}$ が成り立つ．仮定より $\mathsf{ST}(k; A_k, B_k) = \sum_{(u,v)\in E[G^{opt}_{T(k,A_k,B_k)}]} d(u,v)$ であり，また $A_{k+1} = A_k, B_{k+1}\backslash\{v\} = B_k$ に注意すると，$\mathsf{ST}(k+1; A_{k+1}, B_{k+1})$ は式 (10) を表す．

- X_{k+1} が $v \in V$ を forget する場合

  $G_{k+1} = G_k$ より，$G^{opt}_{T(k+1,A_{k+1},B_{k+1})}$ は $G_{T_A} = G^{opt}_{T(k,A_k,B_k)}$ $(v \in A_k)$，$G_{T_B} = G^{opt}_{T(k,A_k,B_k)}$ $(v \in B)$ のうち，総重みの小さい方と等しい．仮定より $v \in A_k$ ならば $\mathsf{ST}(k; A_k, B_k) = \sum_{(u,v)\in E[G^{opt}_{T_A}}$ であり，また $v \in B_k$ ならば $\mathsf{ST}(k; A_k, B_k) =$

$\sum_{(u,v)\in E[G^{opt}_{T_B}}$ である．さらに $v \in A_k$ ならば $A_{k+1} \cup \{v\} = A_k, B_{k+1} = B_k$，$v \in B_k$ ならば $A_{k+1} = A_k, B_{k+1} \cup \{v\} = B_k$ であることに注意すると，$\mathsf{ST}(k+1; A_{k+1}, B_{k+1})$ は式 (10) を表す．

以上より，$i = k+1$ でも式 (11)(12) が式 (10) を表している．数学的帰納法により Lemma 8が証明された． □

最後に Compute の計算量を示す．

Lemma 9. *DAG $G = (V, E)$ の頂点数が n であるとする．このとき，幅が w である G の DAG-PD P，根 $r \in V$，$R \subseteq V$ が与えられたとき，$k = |R|$ とすると，$\mathsf{Compute}(P, r, R)$ は $O(2^w(k+w)n + n^2)$ の計算時間で最適解を出力する．*

Proof. Preprocessing において，r から到達可能な頂点集合の計算に $O(n^2)$ かかる．First Step, Final Step は各々O(1) で計算できる．以下で Exection Step での計算量を考える．各 X_i に対し，$|X_i| \leq w+1$ に注意すると，A_i, B_i の組み合わせは高々2^{w+1} 通り存在する．また Proposition 1より $0 \leq i \leq 2|V|+1$ である．さらに X_i で $v \in V$ を introduce するとき，$\mathsf{pred}(v) \cap A_i \neq \varnothing, \min$ の計算，$v \in R \cup r$ の判定がそれぞれ高々$O(w), O(w), O(k)$ かかることに注意すると，$\mathsf{ST}(i; A_i, B_i)$ の計算量は，X_i が introduce ならば $O(k+w)$ であり，forget ならば $O(1)$ である．よって Exection Step での計算量は $O(2^w(k+w)n)$ である．以上より，Compute の計算量は $O(2^w(k+w)n + n^2)$ である． □

さらに，上記のアルゴリズムの簡単な拡張により，頂点重み付き有向シュタイナー木問題を効率的に解くことができる．

Theorem 5. *頂点重み付き DAG G に対し，ターミナルのサイズ $k = |R|$，幅が w である G の nice DAG-PD が与えられたとき，G の vertex-weighted DST problem を $O(2^w(k+w)n + n^2)$ で解く FPT アルゴリズムが存在する．*

3.5 木分解と比較したときの利点

DAG 上の NP 困難な問題に対し，基礎グラフの木幅を用いてパラメータ化アルゴリズムを構築できる場合がある．しかし木分解は枝の向きの情報をもたないため，アルゴリズムの構築が複雑になることがある．一方，DAG パス分解は枝の向きの情報をもつため，一般に木分解を利用するよりもアルゴリズムの構築が容易になる場合が多い．本節では Directed Edge Dominating Set Problem(DEDS problem) に対する木分解を用いたアルゴリズム [22] と，DAG パス分解を用い

たアルゴリズムとの比較を行い，DAG パス分解の特長を示す．まず DEDS に関する定義と定理を与える．

Definition 21 (Directed Edge Dominating Set)**.** 有向グラフ $G = (V, E)$ に対し，$S \subseteq E$ が G の Directed Edge Dominating Set (DEDS) であるとは，任意の $(v, w) \in E$ に対し，$(v, w) \in S$ であるか，もしくはある $(u, v) \in S$ が存在することの少なくとも一方が成り立つことである．minimum DEDS (mDEDS) とは，全ての DEDS S のうち $|S|$ が最小のものである．

DEDS problem とは，G の mDEDS のサイズを求める問題である．Hakana ら [23] は以下を示した．

Proposition 5 (計算量クラス)**.** *DEDS problem* は出次数が制限された平面的 *DAG* の場合でも *NP* 困難である．

さらに [22] では木幅を用いた FPT アルゴリズムが提案された．

Proposition 6 (木幅を用いた FPT アルゴリズム)**.** 有向グラフ G の基礎グラフの木幅が高々tw であるとき，G の *DEDS problem* を $4^{2tw^2}8^{2tw}n^{O(1)}$ の時間で解く *FPT* アルゴリズムが存在する．

上記のアルゴリズムは一般の有向グラフ上で利用でき，より一般化した問題である (p, q)-DEDS problem に対しても動作するといった利点がある一方，木幅が入力グラフの向きの情報をもたないためアルゴリズムが複雑になっている．これに対し，一般の DAG に対して幅を w とする nice DAG パス分解が与えられたとき，$O(2^{w^2}w^2n^2)$ の時間で DEDS problem を解くアルゴリズムを構築する．木幅が抑えられたグラフでも DAG パス幅が任意に大きくなる場合があるため，計算量の単純な比較はできないが，このアルゴリズムは入力グラフの向きの情報をもつ DAG パス分解を利用するため，アルゴリズムの構築が [22] より単純になるといった利点がある．

Theorem 6. *DAG* G に対し，幅が w である G の *nice DAG-PD* が与えられたとき，G の *DEDS problem* を $O(2^{w^2}w^2n^2)$ で解くアルゴリズムが存在する．

上記のアルゴリズムの構成の前に，有向グラフの Line graph に関する定義を与える．

Definition 22 (有向グラフの Line graph)**.** 有向グラフ $G = (V, E)$ に対し，有向グラフ $L(G) = (V_L, G_L)$ が G の Line graph であるとは，V_L, E_L が以下を満た

すときである．

$$V_L = \{e | e \in E\}$$
$$E_L = \{(e_1, e_2) | e_1 = (u, v) \in E, e_2 = (v, w) \in E, \}$$

すなわち，有向グラフの Line graph とは元のグラフに対して枝の向きを保ちながら頂点と枝を入れ替えたようなグラフである．以下では DAG の Line graph に対する DAG パス分解についての補題を示す．また DAG G の Line graph を $L(G)$ と表す．

Lemma 10. *DAG G に対し，幅が w である G の nice DAG-PD が与えられたとき，幅が高々$O(w^2)$ である $L(G)$ の nice DAG-PD を G の頂点数の多項式時間で構築できる．*

Proof. DAG G に対し，幅が w である G の nice DAG-PD $X = (X_1, X_2, \ldots, X_s)$ が与えられたとする．ここで各 $i = 1, 2, \ldots, s$ に対し，X_i によって誘導される G の部分グラフを G_i とし，$X_i^L = V[L(G_i)]$ とする．このとき $X_L = (X_1^L, X_2^L, \ldots, X_s^L)$ は $L(G)$ の DAG パス分解である．まず X_L は DAG パス分解のルール 1 を満たす．これは DAG パス分解のルール 2 より，G の任意の枝 (u, v) に対して u, v を同時に含むバッグ X_k が X に存在する．これより $(u, v) \in X_k^L$ がいえ，これをすべての枝に対して考えることにより X_L の各バッグの和集合が $V[L(G)]$ と一致することが示される．したがって X_L は DAG パス分解のルール 1 を満たす．また X_L は DAG パス分解のルール 3 も満たす．なぜならば，もしルール 3 を満たさないと仮定すると，ある頂点 $e \in V[L(G)]$ と整数 k, l, m $(1 \leq k < l < m \leq s)$ が存在し，$e \in X_k^L, e \notin X_l^L, e \in X_m^L$ がいえる．このとき $e = (u, v)$ $(u, v \in V[G])$ とすると，$u \in X_k, u \notin X_l, u \in X_m$ もしくは $v \in X_k, v \notin X_l, v \in X_m$ の少なくとも一方が成り立つ．これは X が DAG パス分解のルール 3 を満たすことと矛盾する．したがって X_L は DAG パス分解のルール 3 を満たす．さらに X_L は DAG パス分解のルール 2 も満たす．X において X_i が頂点 v を introduce する場合，DAG パス分解のルール 2 より $\mathsf{pred}(v) \subseteq X_{i-1}, \mathsf{suc}(v) \cap X_{i-1} = \varnothing$ が成り立つ．ここで $\mathsf{pred}(v) = \{u_1, u_2, \ldots, u_k\}$ とすると，X_i^L は X_{i-1}^L に対して $L(G)$ の k 個の頂点 $(u_1, v), (u_2, v), \ldots, (u_k, v)$ が新たに追加されている．各 u_l $(1 \leq l \leq k)$ に対し，u_l の先行頂点 $u_l' \in X_i$ が存在する場合，$(u_l', u_l), (u_l, v) \in X_i^L$ かつ $(u_l, v) \notin X_{i-1}^L$ が成り立つ．これを全ての u_l, u_l' について考えることにより，任意

の枝 $((u'_l, u_l), (u_l, v)) \in E[L(G)]$ に対して X_L は DAG パス分解のルール 2 を満たすことがいえる．X_i が頂点 v を forget する場合，X_i^L では v を端点にもつ枝に対応する L(G) 上の頂点が削除されるだけであるため，DAG パス分解のルール 2 に反しない．以上より X_L は $L(G)$ の DAG パス分解であることが示される．このとき X^L の幅は高々$O(w^2)$ である．最後に X_L の nice DAG パス分解 X'_L を計算することにより目的の DAG パス分解を得る．計算量については，各バッグ X_i^L で新たに追加される頂点数は高々w 個であり，それぞれに対して先行頂点が存在するかの判定に高々$O(w)$ かかる．バッグ数は Proposition 1より，$n = |V[G]|$ とすると $O(|V[L(G)]|) = O(n^2)$ である．nice DAG パス分解の計算も n の多項式時間で行えることに注意すると，X から X'_L の構築は n の多項式時間で行えることが示される． □

一方，DAG G の mDEDS と $L(G)$ の mDiDS との関係について，以下の補題を示す．

Lemma 11. *DAG G に対し，G の mDEDS と $L(G)$ の mDiDS は等しい．*

Proof. $S \in E[G]$ が G の DEDS であるとき，任意の枝 $(v, w) \in E[G]$ について $(v, w) \in S$ であるか，ある枝 $(u, v) \in S$ が存在するかのいずれかが成り立つ．このとき $L(G)$ 上では任意の頂点 $(v, w) \in V[L(G)]$ について $(v, w) \in S$ であるか，ある頂点 $(u, v) \in S$ が存在するかのいずれかが成り立つ．したがって S は $L(G)$ の DiDS である．同様の議論により $S \in V[L(G)]$ が $L(G)$ の DiDS ならば S は G の DEDS である．したがって G の DEDS と $L(G)$ の DiDS は一対一で対応する．よって G の mDEDS と $L(G)$ の mDiDS は等しいことが示される． □

上記の 2 つの補題を使うことで Theorem 6を示す．

Proof. Lemma 11より，$L(G)$ の mDiDS を求めればよい．また Lemma 10より，幅が w である G の nice DAG-PD X が与えられたとき，幅が高々$O(w^2)$ である $L(G)$ の nice DAG-PD X'_L を多項式時間で構築できる．よって 3.1節のアルゴリズム Compute(X'_L) を計算することで $L(G)$ の mDiDS を求めることができる．このとき X'_L の幅が $O(w^2)$，バッグ数が $O(n^2)$ となっていることに注意すると，計算量は $O(2^{w^2} w^2 n^2)$ である． □

第4章　DAGパス分解を求める$O(\log^2 n)$-近似アルゴリズム

本節では頂点数n，最大出次数k，パス幅pwであるDAG Gに対し，幅が高々$O(k \cdot pw \log^2 n)$であるDAGパス分解を与えるアルゴリズムを設計する．

4.1　定義と補題

まず最小化問題に対する近似率の定義を示す．

Definition 23 (近似率)**.** 問題の入力Iに対し，アルゴリズムが出力する解をS_{alg}，最適解をS_{opt}，目的関数の値を$f(S)$とする．このとき近似率rは以下のように定義される．

$$r = \sup_I \frac{f(S_{alg})}{f(S_{opt})}$$

次にseparatorに関する定義と補題を与える．

Definition 24 (DAG edge-separator)**.** [19] 各枝がコストを持つDAG Gに対し，DAG edge-separator Sとは，Gの頂点をA, Bの2つに分割する枝集合であり，任意の$e \in S$はAから出てBに入る．Sのコストとは，全ての$e \in S$のコストの総和であり，$C(S)$と表す．

以下では枝のコストを全て1とする．すなわち枝集合Eのコストは$|E|$となる．

Definition 25 (b-balanced DAG edge-separator)**.** b-balanced DAG edge-separatorとは，各A, Bについて$|A| \geq b|V(G)|, |B| \geq b|V(G)|$を満たすDAG edge-separatorである．minimum b-balanced DAG edge-separatorとは，全てのb-balanced DAG edge-separator Sのうち，$C(S)$が最小のものである．

Raviらは[19]で以下の補題を示している．

Lemma 12. *頂点数nのDAG Gが与えられたとき，Gの minimum $\frac{1}{3}$-balanced DAG edge-separator Sのコスト$C(S)$に対し，コスト$O(C(S) \log n)$の$\frac{1}{8}$-balanced DAG edge-separatorを見つける多項式時間アルゴリズムが存在する．*

上記のアルゴリズムを利用して，出次数が制限されたDAG Gに対し，パス幅の$O(\log^2 n)$倍以内の幅を持つDAG-PDを与える近似アルゴリズムを構成する．

4.2 $O(\log^2 n)$-近似アルゴリズム

Theorem 7. 頂点数 n，最大出次数 k，*DAG-pathwidth* が pw の *DAG* G が与えられたとき，幅が $O(k\ pw \log^2 n)$ の *DAG-PD* を与える多項式時間アルゴリズムが存在する．

上記のアルゴリズムを構成するため，まず関数 Merge を定義する．Merge は，DAG G が G_L と G_R に分割されているとき，G_L, G_R のそれぞれの nice DAG-PD である P_L, P_R を受け取り，G の nice DAG-PD P を返す関数である (Figure**??** を参照)．このとき，G_L, G_R の間の枝の向きは全て G_L から出て G_R に入る向きであることに注意する．

Subroutine 1 (Merge)**.** G_L から出て G_R に入る枝の端点のうち，G_L に含まれる頂点の集合を V' とする．Merge(P_L, P_R) では，$P_L = (X_1, X_2, \ldots, X_{s_l}), P_R = (Y_1, Y_2, \ldots, Y_{s_r})$ に対し，以下の操作を全ての $v \in V'$ に対して行ってできる *nice DAG-PD* P'_L, P'_R を構成する．

- v が P_L で *forget* されたバッグを X_i とし，各 X_j $(j = i, i+1, \ldots, s_p)$ に v を追加する．
- v を各 Y_k $(k = 1, 2, \ldots, s_r)$ に追加する．

こうしてできた $P'_L = (X'_1, X'_2, \ldots, X'_{s_l}), P'_R = (Y'_1, Y'_2, \ldots, Y'_{s_r})$ に対し，$P = (X'_1, X'_2, \ldots, X'_{s_l}, Y'_1, Y'_2, \ldots, Y'_{s_r})$ を出力する．

.

Lemma 13. $P =$ Merge(P_L, P_R) は G の *nice DAG-PD* である．

Proof. P について，$X'_1 \cup X'_2 \cup \cdots \cup X'_{s_l} \cup Y'_1 \cup Y'_2 \cup \cdots \cup Y'_{s_r} = V(G)$ であるため，DAG-PD のルール 1 を満たす．またルール 3 について，任意の $v \in V'$ がルール 3 を満たしていることを示せば十分．v が P_L で introduce されたバッグを X_m とすると，Merge の操作により，v は P において $X'_m, X'_{m+1}, \ldots, Y'_{s_r}$ のバッグにのみ含まれ，それ以外のバッグには含まれない．よってルール 3 を満たす．ルール 2 について，各 $v \in V'$ とそれに接続する各頂点 $u \in G_R$ に対して，(v, u) がルール 2 を満たすことを示せば十分．v, u がそれぞれ P_L, P_R で introduce されたバッグを X_m, Y_n とすると，v は P において $X'_m, X'_{m+1}, \ldots, Y'_{s_r}$ のバッグにのみ含まれ，u は $Y'_n, Y'_{n+1}, \ldots, Y'_{s_r}$ のバッグにのみ含まれる．したがって $v, u \in Y'_n$ かつ $u \notin Y'_{n-1}$ が成り立ち，ルール 3 を満たす． □

次に，DAG G が与えられたときに G の nice DAG-PD を出力するアルゴリズ

ム Make を示す.

Make($G[V]$)

1. Termination Step: $|V| = 1$ ならば，G の自明な nice DAG-PD $P = (\varnothing, V, \varnothing)$ を返す.
2. Divide Step: A, B を，Lemma 12のアルゴリズムによる G の分割とする．再帰的に $D_A = \mathsf{Make}(G[A]), D_B = \mathsf{Make}(G[B])$ を計算する.
3. Combine Step: $D = \mathsf{Merge}(G[A], G[B])$ とし，D を返す.

ここで Theorem 7の証明のため，次の補題を示す.

Lemma 14. 頂点数 n，最大出次数 k，*DAG-pathwidth* pw である *DAG* G に対し，コストが高々$k(pw+1)$ の *minimum* $\frac{1}{3}$*-balanced DAG edge-separator* が存在する.

Proof. $G = (V, E)$ の最適な nice DAG-PD を $P = (X_1, X_2, \ldots, X_s)$ とする．P の DAG-pathwidth は pw である．G の各頂点を P で introduce された順に $v_1, v_2, \ldots, v_n$ とする．n が偶数のとき，$v = v_{\frac{n}{2}}$，n が奇数のとき，$v = v_{\frac{n+1}{2}}$ とし，v が初めて introduce されたバッグを X_i とする．$A = X_1 \cup X_2 \cup \cdots \cup X_i, B = X_{i+1} \cup X_{i+2} \cup \cdots \cup X_s$ とする．例として Fiture**??**を示す．このとき Xi から出て B に入る枝の集合を S とすると，DAG-PD のルール 3 より $G' = (V, E \backslash S)$ には A と B の間に枝はなく，v の選び方により $|A| \geq \frac{1}{3}|V|, |B| \geq \frac{1}{3}|V|$ を満たす．また DAG-PD のルール 2 より A と B の間の枝は A から出て B に入る向きである．よって S は G の $\frac{1}{3}$-balanced DAG edge-separator である．G の最大出次数が k であることに注意すると，S のコスト $C(S)$ について以下の不等式が成り立つ.

$$C(S) \leq k|X_i| \leq k(pw+1)$$

一方，G の minimum $\frac{1}{3}$-balanced DAG edge-separator S_{opt} のコストは高々$C(S)$ であるから，

$$C(S_{opt}) \leq k(pw+1)$$

□

最後に次の補題を示すことで，Theorem 7が成り立つことを示す.

Lemma 15. 頂点数 n，最大出次数 k，*DAG-pathwidth* が pw の *DAG* $G[V]$ が与えられたとき，$P = \mathsf{Make}(G[V])$ とする．このとき，ある定数 α が存在し，P の幅は高々 $\alpha k(pw+1)\log^2 n$ である．

Proof. DAG G_L, G_R があり，G_L, G_R の間の枝は，G_L から出て G_R に入る向きであるとする．また G_L, G_R に対して，ある nice DAG-PD P_L, P_R があり，幅がそれぞれ w_l, w_r であるとする．ここで G_L, G_R の間の枝の端点のうち，G_L に含まれる頂点集合を V' とする．このとき $\mathsf{Merge}(P_L, P_R)$ の動作中に構成される P'_L, P'_R の幅は，それぞれ高々 $w_l+|V'|, w_r+|V'|$ である．なぜならば，P'_L は P_L の各バッグに高々 $|V'|$ 個の頂点を追加する操作を行い，P'_R は P_R の各バッグにちょうど $|V'|$ 個の頂点を追加する操作を行うからである．したがって $\mathsf{Merge}(P_L, P_R)$ によって得られる nice DAG-PD P_{LR} の幅 w_{lr} は，

$$\begin{aligned} w_{lr} &\leq \max(w_l + |V'|, w_r + |V'|) \\ &= \max(w_l, w_r) + |V'| \end{aligned}$$

G_L, G_R の間の枝数を m とすると，$|V'|$ は高々 m であるから，

$$w_{lr} \leq \max(w_l, w_r) + m$$

一般性を保って $w_l \leq w_r$ とすると，

$$w_{lr} \leq w_r + m$$

minimum $\frac{1}{3}$-balanced DAG edge-separator のコストを m_{opt} とすると，Lemma 12 より，ある定数 α があり，$m \leq \alpha m_{opt} \log n$ とできるため，

$$w_{lr} \leq w_r + \alpha m_{opt} \log n$$

Make は Merge を高々 $\log n$ 回繰り返すため，Make が最初に構成するサイズ 1 の nice DAG-PD の幅が 0 であることに注意すると，Make が出力する nice DAG-PD の幅 w は，

$$\begin{aligned} w &\leq 0 + (\alpha m_{opt} \log n) \log n \\ &= \alpha m_{opt} \log^2 n \end{aligned}$$

Lemma 14より，$m_{opt} \leq k(pw+1)$ であるから，

$$w \leq \alpha k(pw+1)\log^2 n$$

以上より，P の幅は高々$\alpha k(pw+1)\log^2 n$ である． □

4.3 $O(\log^{3/2} n)$-近似アルゴリズム

上記のアルゴリズムにより $O(\log^2 n)$ の近似率である幅を持つ DAG パス分解を得られるが，以下では one-shot BP との等価性を示すことでさらに $O(\log^{3/2} n)$ の近似率が達成できることを示す．

Theorem 8. 頂点数 n，*DAG-pathwidth* が pw の *DAG* G が与えられたとき，幅が $O(pw \cdot \log^{3/2} n)$ の *DAG-PD* を与える多項式時間アルゴリズムが存在する．

Per ら [20] は one-shot BP の近似アルゴリズムの存在を示している．

Lemma 16. 頂点数 n の *DAG* G に対し，*one-shot BP* のペブリング数が最小値の高々$O(\log^{3/2} n)$ 倍である戦略を出力するアルゴリズムが存在する．

したがって，Theorem 8を示すためには以下の補題を示せば十分である．

Lemma 17. *one-shot BP* と *nice DAG-PD* は等価な問題である．

Proof. まず DAG G に対し，X が G の nice DAG-PD であれば X は G の one-shot BP であることを示す．G に対するある nice DAG-PD を $X = (X_1, X_2, \ldots, X_s)$ とする．ルール 4 より $X_i = \varnothing$ であるため one-shot BP の初期条件を満たす．ルール 1 とルール 3 より，すべての頂点は X でちょうど 1 回 introduce されるため，one-shot BP ですべての頂点がちょうど 1 回 pebble されるという条件を満たす．操作 1 とルール 2 より，$v \in V$ が X_i で introduce されるとき，任意の $(u, v) \in E$ に対して $u \in X_{i-1}$．これは one-shot BP の pebble のルールを満たす．また各頂点に対し，forget がちょうど 1 度だけ行われることに注意すると，操作 2 と unpebble の操作は明らかに等価である．よって X の introduce と forget が各々pebble と unpebble に対応する．以上より X は G の one-shot BP である．

次に P が G の one-shot BP であれば P は G の nice DAG-PD であることを示す．G に対するある one-shot BP を $P = (P_1, P_2, \ldots, P_t)$ とする．$P_1 = \varnothing$ より nice DAG-PD のルール 4 を満たす．また各頂点 $v \in V$ に対し，v は P でちょうど 1 回ずつ pebble，unpebble されるため，nice DAG-PD のルール 1 を満たす．また，v の pebble，unpebble がそれぞれ P_i, P_{k+1} $(1 \leq i \leq k \leq t-1)$ で行われ

るとすると，任意の頂点に対して 2 回以上 pebble されることはないため，任意の P_j $(i \leq j \leq k)$ において $v \in P_j$ を満たす．これを V の全ての頂点に対して考えることで，nice DAG-PD のルール 3 を満たす．さらに pebble のルールより，$v \notin P_{i-1}$ かつ任意の $(u, v) \in E$ に対し $u \in P_{i-1}$ ならば $P_i = P_{i-1} \cup \{v\}$ であるが，これは $u, v \in P_i$ かつ $v \notin P_{i-1}$ を満たす．したがって pebble の操作は nice DAG-PD のルール 2 を満たした introduce の操作とみなすことができる．また unpebble の操作は明らかに nice DAG-PD の forget の条件を満たす．以上より P は G の nice DAG-PD である． □

第5章　$O(ld^t)$の幅のDAGパス分解を求めるアルゴリズム

本節では最大出次数d，根の個数lであるDAG Hと非負整数tが与えられたとき，$O(ld^t)$の幅を持つDAG-PDを出力するか，Hのパス幅がtよりも大きいことを示す証拠を与えるFPTアルゴリズムを提案する．

5.1　無向グラフのパス分解を求めるアルゴリズム

Kevinら[8]は以下の補題を示している．

Lemma 18. 頂点数nの無向グラフH，整数tが与えられたとき，Hのパス幅がtより大きい証拠を与えるか，幅が高々$O(2^t)$のパス分解を与えるような$O(n)$の計算時間のアルゴリズムが存在する．

上記のアルゴリズムを参考にし，最大出次数と根数がそれぞれd, lであるDAG Hに対し，Hのパス幅がtより大きい証拠を与えるか，幅が高々$O(ld^t)$のパス分解を与えるような多項式時間FPTアルゴリズムを構成する．

5.2　アルゴリズムの構築

アルゴリズムでは以下のembeddingを利用する．

Definition 26 (embedding)**.** 有向グラフ$G_1 = (V_1, E_1)$の，有向グラフ$G_2 = (V_2, E_2)$へのembeddingとは，V_1からV_2への単射であり，かつ任意の枝$(u, v) \in E_1$からE_2の有向辺素パスへの写像が存在することである．

本節では一般のDAGに対して以下が成り立つことを示す．

Theorem 9. *Hを最大出次数と根数がそれぞれd, lである任意のDAG，tを非負整数とする．このとき以下のいずれか一方が必ず成り立つ．*

(a) *HのDAGパス幅は高々$ld^{t+3} - 1$である．*

(b) *Hは2つのDAG A, Bに分割できる．このときAとBの間にはAからBへの枝しか存在せず，Aのパス幅はtより大きく$ld^{t+3} - 1$より小さい．*

Theorem 9を示す前に，以下の補題を示す．

Lemma 19. $T_{h,d}$を高さhの完全有向d分木$(h, d > 1)$とする．このとき$T_{h,d}$

の *DAG* パス幅は $h-1$ である．

Proof. h に関する数学的帰納法で示す．$h=2$ のとき $T_{2,d}$ のパス幅は明らかに 1 であるため補題を満たす．ここである $h>1$ での補題の成立を仮定する．仮定より幅が $h-1$ であるような DAG-PD X_h が存在する．ここで $T_{h+1,d}$ は単一の根 r から d 個の $T_{h,d}$ の根に枝を伸ばしたグラフであることに注意すると，$T_{h+1,d}$ の最適なパス分解は，はじめに r の根をバッグに含め，以降で X_h の各バッグを順に d 回分つなげたものである．$h,d>1$ に注意すると，このとき明らかに ($T_{h+1,d}$ の DAD パス幅) $=1+(h-1)=h$ が成り立つため補題を満たす．以上より 2 以上の任意の h,d に対して補題が成り立つ． □

以下では．実際にアルゴリズムを構築することで Theorem 9を示す．

Proof. (Theorem 9)
入力グラフ $H=(V,E)$ に対し，高さが $\lceil \log_d l \rceil$ である完全有向 d 分技 (ただし根自身の高さを 1 とする) を用意し，出次数が高々d となるように完全有向 d 分木の葉から H の各根に向かう枝を加えて結合する．こうしてできるグラフを H' とする．また $M_{t,d,l}$ を高さ $\lceil \log_d l \rceil + t + 2$ の完全有向 d 分木とする．

以下のアルゴリズムでは $M_{t,d,l}$ の H' への embedding を見つける．もし embedding を見つけることができれば H' の DAG パス幅は $M_{t,d,l}$ の DAG パス幅以上であることがわかる．embedding の探索では部分的な embedding を行い，$M_{t,d,l}$ の頂点をトークンと呼ぶ．token T が H' のある頂点に置かれたとき T は tokened であると表し，頂点に置かれていないとき untokened であると表す．アルゴリズムを通して H' の 1 つの頂点に置くことができるトークンは 1 つのみである．

以下で各トークンの再帰的なラベル付けを与える．図**??**で H に対する $H', M_{t,d,l}$ の構成例を示す．

1. 根のトークンは空文字列 λ とラベル付けする．
2. 高さ h の親トークン $m=\lambda b_1 b_2 \dots b_{h-1}$ の子トークンを左から順に $m\cdot 1, m\cdot 2, \dots, m\cdot d$ とラベル付けする．

時刻 i においてトークンが置かれた H' の頂点集合を X_i とする．頂点集合の列 $(X_1, X_2, \dots, X_s)$ は H' のパス分解，もしくは H' の分割で Theorem 9の条件を満たす A' のパス分解を表す．

アルゴリズムの初期状態では H' のすべての頂点が blue で塗られていると仮

Algorithm 1 GrowTokenTree

1: **while** there is a vertex $u \in H'$ with token T and a blue neighbor v whose all predecessors are placed token, and token T has an untokened child $T \cdot b$ **do**
2: place token $T \cdot b$ on v
3: **end while**
4: **return** {tokened vertices of H'}

定する．アルゴリズムの進行中，H' のある頂点 v にトークンが置かれた場合，v の色を red に変更し，トークンが v から削除されても red のままであるとする．このとき blue の頂点にのみトークンを置くことができるため，H' の各頂点は高々一度しかトークンを置くことができない．

アルゴリズム FindEmbedding は，(1) 列 4 で H' が blue の頂点を持たなくなったときか，(2) 列 4 で $|X_i| = |V[M_{t,d,l}]|$ が成り立つときか，(3) 列 14 の処理が行われたときのいずれかで終了する．以下では (1)(2)(3) のそれぞれの場合においてアルゴリズムが正しく動作することを示す．

まず (1) の場合を考える．FindEmbedding の列 4 において $i = s$ の時点で H' のすべての頂点が red になって終了した場合，アルゴリズムが出力する頂点集合の列 $X_{H'} = (X_1, X_2, \ldots, X_s)$ は H' の DAG-PD になっていることを示す．まず任意の頂点 $v \in H'$ は red であるためいずれかの頂点集合 X_i に必ず含まれる．よって DAG-PD のルール 1 を満たす．また v はちょうど 1 度だけ blue から red に変わり，red のときにのみトークンが削除されてそれ以降 token が置かれることはないため，v を含む頂点集合 X_i は連結なパスを構成する．したがって DAG-PD のルール 3 を満たす．さらに任意の枝 $(u, v) \in E[H']$ について v が blue から red に変わる時点を i とする．このとき FindEmbedding の列 5 の条件より i より前の時点で u から token は削除されず，また GrowTokenTree の while の条件より，v の先行頂点にはすべて token が置かれているため，u もまた X_i に含まれる．v は $X_{i'}$ $(i' < i)$ に含まれないことに注意すると $u, v \in X_i, v \notin X_{i-1}$ が成り立つため DAG-PD のルール 3 を満たす．以上より頂点集合の列 $X_{H'}$ は H' の DAG-PD となっている．$\lceil \log_d l \rceil < \log_d l + 1$ に注意すると，このとき

Algorithm 2 FindEmbedding

place root token λ on root of H'
$i \leftarrow 1$
$X_i \leftarrow$ call GrowTokenTree
while $|X_i| < |V[M_{t,d,l}]|$ and H' has at least one blue vertex **do**
  if there is a vertex $v \in H'$ with token T that v has no blue successor and T has at most one tokened child **then**
    remove T from H'
    if T had one tokened child $T \cdot b$ **then**
      replece $T \cdot b$ with T on H'
    end if
    while T had a tokened grandchild $T \cdot b_1 \cdot b_2$, and $T \cdot b_2$ is untokened **do**
      replece $T \cdot b_1 \cdot b_2$ with $T \cdot b_2$ on H'
    end while
  else
    return X_i
  end if
  $i \leftarrow i + 1$
  $X_i \leftarrow$ call GrowTokenTree
end while

$X_{H'}$ の幅は高々$|V[M_{t,d,l}]| = d^{\lceil \log_d l \rceil + t + 2} - 1 < ld^{t+3} - 1$ である．ここで $X_H = (X_1 \cap V[H], X_2 \cap V[H], \ldots, X_s \cap V[H])$ は H に対して DAG-PD の 3 つのルールを満たすことから H の DAG-PD になっていることに注意すると，X_H の幅もまた高々$ld^{t+3} - 1$ である．したがって幅が高々$ld^{t+3} - 1$ である H の DAG-PD が得られる．

次に (2) の場合を考える．FindEmbedding の列 4 において $i = s$ の時点で $|X_s| = |V[M_{t,d,l}]|$ が成り立つとき，頂点集合 $X_1 \cup X_2 \cup \cdots \cup X_s$ によって誘導される H' の部分グラフを A' とする．このときアルゴリズムが出力する頂点集合の列

$X_{A'} = (X_1, X_2, \ldots, X_s)$ は A' の DAG-PD になっている．まず A' の定義より明らかに DAG-PD のルール 1 を満たす．また上記と同様の議論により DAG-PD のルール 2, 3 を満たす．よって $X_{A'}$ は A' の DAG-PD である．ここで A を頂点集合 $V[A'] \cap V[H]$ によって誘導される部分グラフとし，$X_A = (X_1 \cap V[H], X_2 \cap V[H], \ldots, X_s \cap V[H])$ とする．X_A は A に対して DAG-PD の 3 つのルールを満たすことから A の DAG-PD になっていることに注意すると，上記と同様の議論により幅が高々 $ld^{t+3} - 1$ である A の DAG-PD が得られる．

さらに，$X_{A'}$ の右端のバッグ X_s によって誘導される H' の部分グラフ $H_{A'3}$ は，$M_{t,d,l}$ の H' への embedding を表していることを示す．GrowTokenTree では $u, v \in H'$ にそれぞれトークン $T, T \cdot b \in M_{t,d,l}$ を配置する操作なので，明らかに embedding の条件を満たす．したがって FindEmbedding の列 6 から列 12 において embedding の条件が満たされることを示せば十分である．列 5 の処理の時点で $M_{t,d,l}$ の各頂点に対して embedding の条件が満たされていると仮定する．列 5 の条件を満たすトークン T が存在し，tokened な子を 1 つだけ持つとき，列 6 と列 8 により $T \cdot b$ は T に置き換えられる．このとき T の親 T' と $T, T \cdot b$ のそれぞれ置かれていた頂点を $u, v, w \in H'$ とすると，置き換えによって w に T が置かれるが，仮定より枝 (T', T) から辺素パス $u \to v$ への写像があるため，T はただ一つの tokened な子 $T \cdot b$ をもつことに注意して，明らかに枝 (T', T) から辺素パス $u \to w$ への写像が存在する．したがって embedding の条件は満たされる．T が tokened な子を持たないときは列 6 のみ実行され，列 8 は実行されないが，T を削除しても明らかに embedding の条件は満たされる．次に列 10 から 12 の処理を考える．GrowTokenTree では $M_{t,d,l}$ の根に近いトークンから順に tokened になるため，T が tokened な子を持たないならば T の孫もまた tokened でない．したがって列 11 は行われない．T がただ一つの tokened な子 $T \cdot b_1$ を持つ場合，GrowTokenTree の操作と T が tokened な子を持たない場合の操作より，T の tokened な孫は必ず $T \cdot b_1$ を親に持つ．ここで列 8 の処理により $T \cdot b_1$ が untokened になるため T のすべての子が untokened となるが，列 11 の処理により tokened な T の孫 $T \cdot b_1 \cdot b_2$ はすべて T の子 $T \cdot b_2$ に置き換えられる．このとき $T \cdot b_1, T \cdot b_1 \cdot b_2$ が置かれていた頂点をそれぞれ $u, v \in H'$ とすると，仮定より $M_{t,d,l}$ の枝 $(T \cdot b_1, T \cdot b_1 \cdot b_2)$ から H' の辺素パス $u \to v$ への写像があるため，トークンの置き換え後は明らかに枝 $(T, T \cdot b_2)$ から辺素パス $u \to v$ への写像が存在する．したがって embedding の条件は満たされる．

以上より $X_{A'}$ が出力されるとき，A' は $M_{t,d,l}$ の H' への embedding を表している．ここで A' は高さ $\lceil \log_d l \rceil + t + 2$ の完全有向 d 分木を含み，頂点集合 $V[A']\backslash V[A]$ によって誘導される部分グラフは高さが高々 $\lceil \log_d l \rceil$ の完全有向 d 分木であることに注意すると，A は少なくとも $(\lceil \log_d l \rceil + t + 2) - (\lceil \log_d l \rceil) = t + 2$ の高さをもつ完全有向 d 分木 T_A を含む．すなわち T_A から A への embedding が存在する．Lemma 19より T_A のパス幅は $t+1$ であるため，A のパス幅もまた $t+1$ 以上である．よって A を内部に含む H のパス幅は t より大きいことが示される．

最後に (3) の場合を考える．FindEmbedding の列 14 の処理が行われたときに A のパス幅が t より大きいことを示す．ここで単一の根 r_G をもつ任意の DAG G の各頂点に対し，r_G からの最長パスの長さが $k-1$ である頂点集合を $L_{G,k}$ と表す．以下では k を G の階層と呼ぶ．ただし $L_{G,1} = \{r_G\}$ とする．H' の各頂点の出次数は高々 d であるため，ある時点 i において $\lambda \in M_{t,d,l}$ が置かれた頂点 r' の階層を $k_{r'}$，頂点集合 $\bigcup_{k_{r'} \le k \le k_{r'} + \lceil \log_d l \rceil + t + 1} L_{H',k}$ によって誘導される部分グラフを $H_{k_{r'},t}$ とすると，$H_{k_{r'},t}$ に含まれる任意の頂点にはトークンを置くことができる．ここで列 14 の処理が行われるとき，任意の頂点 $v \in V[H_{k_{r'},t}]$ は以下の少なくとも一方が成り立つ．

1. ある $w \in \mathsf{suc}(v)$ が存在し，$w \in L_{H,k'}$ $(k' > k_{r'} + \lceil \log_d l \rceil + t + 1)$ を満たし，かつ w は blue である．
2. v に置かれたトークン T が 2 つ以上の tokened な子をもつ．

また，このとき $M_{t,d,l}$ において tokened であるトークンのパス $\lambda \cdot m_1 \cdot m_2 \cdot \ldots m_{\lceil \log_d l \rceil + t + 1}$ が少なくとも 1 つ存在する．なぜならば，もしそのようなパスが存在しなければ，上記の議論により $H_{k_{r'},t}$ は階層が高々 $\lceil \log_d l \rceil + t + 1$ しかなく，$H_{k_{r'},t}$ のすべての頂点にトークンを置くことができ矛盾するからである．ここで $P = \lambda \cdot m_1 \cdot m_2 \cdot \ldots m_{\lceil \log_d l \rceil + t + 1}$ とすると，各トークン $m_i \in P$ (ただし λ を m_0 と表現する) について m_i が置かれた頂点 v_i は上記の条件 1, 2の少なくとも一方を満たす．条件 1を満たすとき，v_i は $v_{\lceil \log_d l \rceil + t + 1}$ にトークン $m_{\lceil \log_d l \rceil + t + 1}$ が置かれるまでは A の DAG-PD において forget することはできない．なぜならば DAG-PD のルール 2 より，条件 1での w の introduce は $v_{\lceil \log_d l \rceil + t + 1}$ の introduce の後，すなわちトークン $m_{\lceil \log_d l \rceil + t + 1}$ の $v_{\lceil \log_d l \rceil + t + 1}$ への配置後であり，w を後続頂点にもつ v_i はそれよりも前に forget できないからである．このとき $u_i = v_i$ とする．m_i が条件 1を満たさず 2を満たすとき，m_i は tokened な子 m'_{i+1} $(\neq m_{i+1})$

を少なくとも 1 つもつ．m'_{i+1} が置かれた頂点についても条件 1，2の少なくとも一方を満たすため，上記の議論を繰り返すことにより，$M_{t,d,l}$ の tokened な葉は条件 1を満たすことに注意すると，m'_{i+1} を根とする有向木には条件 1を満たすような頂点が少なくとも 1 つ存在する．このような頂点を u_i とする．上記と同様の理由により，u_i は $v_{\lceil \log_d l \rceil + t + 1}$ に $m_{\lceil \log_d l \rceil + t + 1}$ が置かれるまでは A' の DAG-PD において forget することができない．これをすべての $m_i \in P$ に対して考えることにより，各頂点 $u_0, u_1, \ldots, u_{\lceil \log_d l \rceil + t + 1}$ を得る．このとき A' の任意の DAG パス分解 X' において $v_{\lceil \log_d l \rceil + t + 1}$ の introduce を行うバッグを X_p とすると，DAG パス分解のルール 2 に注意して X' のバッグ X_1 から X_p までの間では少なくとも各 v_i $(0 \leq i \leq \lceil \log_d l \rceil + t + 1)$ の introduce が行われており，かつ先程の議論により X_p には各 i に対して v_i から u_i までのパス上の頂点が少なくとも 1 つずつ含まれることがいえる．したがって $|X_p| \geq |P| = \lceil \log_d l \rceil + t + 2$ より，A' のパス幅は少なくとも $\lceil \log_d l \rceil + t + 1$ 以上である．ここでトークン列 P のうち A の頂点に置かれたトークンのみを取り出してできるトークン列を P_A とする．$|P \backslash P_A| \leq \lceil \log_d l \rceil$ より $|P_A| = |P| - |P \backslash P_A| \geq (\lceil \log_d l \rceil + t + 2) - \lceil \log_d l \rceil = t + 2$ に注意すると，上記と同様の議論により A のパス幅は t より大きいことが示される．□

Corollary 1. 最大出次数 d，根の個数 l である *DAG* H，整数 t が与えられたとき，H の *DAG* パス幅が t より大きい証拠を与えるか，幅が高々$O(ld^t)$ の *DAG* パス分解を与えるような $O(n^2)$ 時間のアルゴリズムが存在する．

Proof. Theorem 9より，FindEmbedding は H の DAG パス幅が t より大きい証拠として完全 d 分木の H への embedding を出力するか，幅が高々$O(ld^t)$ のパス分解を出力する．以下では FindEmbedding が多項式時間で終了することを示す．GrowTokenTree では列 1 の while の条件の判定は高々$O(dn)$ かかり，これを高々 $|V[M_{t,d,l}]| = O(d^t)$ 回繰り返す．また FindEmbedding の列 6 の T の削除によって，T が置かれていた頂点は red のままであるため，この操作は高々$O(n)$ 回行われることに注意すると，列 4 の while もまた高々$O(n)$ 回行われる．さらに列 10 の while は明らかに高々d^2 回繰り返す．他のステップは $O(1)$ の計算時間で処理される．以上よりアルゴリズムは d, l がある定数以下であるとすると $O(n^2)$ で停止する．□

第6章　DAG-treewidth

6.1　DAG-treewidthの定義

本節では，DAG-PDをさらに一般化した概念であるDAG Tree Decomposition(DAG-TD) を導入する．

Definition 27. 有向グラフ$G = (V, E)$のDAG Tree Decomposition (DAG-TD)とは，有向木Tと，Tの各ノードtについて$\beta(t) \subseteq V$であるようなβとのペア(T, β)である．(T, β)は以下の3つを満たす．

1. $\sum_{t \in T} \beta(t) = V$
2. Tの根をrとしたとき，任意の枝$(u, v) \in E$について以下のいずれかが成り立つ．
   - $u, v \in \beta(r)$
   - あるi $(i \in T, i \neq r)$があり，iの親をjとしたとき，$u, v \in \beta(i), v \notin \beta(j)$
3. 任意の$i, j, k \in T$に対し，jがiからkまでのパス上に存在するならば，$\beta(i) \cap \beta(k) \subseteq \beta(j)$．すなわち各$v \in V$について，$v$を含むバッグは$T$上で非空な部分有向木を誘導する．

DAG-TDの例をFigure**??**で示す．以下でDAG-treewidthを定義する．

Definition 28 (DAG-treewidth)**.** あるDAG-TD (T, β)のwidthを$\max_{t \in T} |\beta(t)|$と定義する．このとき，有向グラフ$G$のDAG-treewidthとは，$G$に対する全てのDAG-TDのうち，widthが最小であるDAG-TDのwidthである．

以降では，GのDAG-pathwidth, DAG-treewidthをそれぞれ$\mathsf{pw}(G), \mathsf{tw}(G)$と表す．

6.2　nice DAG-TD

本節では，nice DAG-PDと類似の概念であるnice DAG-TDを説明する．

Definition 29. $G = (V, E)$を有向グラフとし，(T, β)をGのDAG-TDとする．(T, β)が以下の2つを満たすとき，(T, β)はGのnice DAG-TDであるという．

1. rをTの根としたとき，$\beta(r) = \varnothing$
2. 各$i \in T$に対し，$\beta(i)$は以下のいずれかである．

   introduce　iがただ1つの子jをもち，ある強連結成分Sがあり，$S \cap \beta(j) \neq \varnothing$かつ$\beta(i) = \beta(j) \cup S$

   forget　iがただ1つの子jをもち，ある$v \in V$があり，$\beta(i) = \beta(j) \backslash \{v\}$

union　i がちょうど 2 つの子 j, k をもち，$\beta(i) = \beta(j) = \beta(k)$

nice DAG-TD の例を Figure**??**で示す．ある DAG-TD が与えられたとき，width が同じである G の nice DAG-TD を効率よく構築することができる．

Theorem 10. (T, β) が有向グラフ $G = (V, E)$ の *DAG-TD* であるとする．また (T, β) の *width* を w とする．このとき，*width* が w であるような G の *nice DAG-TD* を多項式時間で構築できる．

Proof. (T, β) に対し，以下の手順を行い nice DAG-PD を得る．

1. T に根 r $(\beta(r) = \varnothing)$ を追加する．
2. ある $i \in T$ がただ 1 つの子 $j \in T$ をもち，かつ $\beta(i) = \beta(j)$ ならば，T から j を除き，i から j のそれぞれの子への枝を加える．このような i, j がなくなるまでこの操作を繰り返す.
3. 以下の 5 つの状態がなくなるまで操作を繰り返す.
   (a) ある $i \in T$ と，その子 $j \in T$ について，$\beta(i) \nsubseteq \beta(j)$ かつ $\beta(j) \nsubseteq \beta(i)$ ならば，i と j の間に i' $(\beta(i') = \beta(i) \cap \beta(j))$ を加え，(i, i', j) の順につなぐ.
   (b) ある $i \in T$ と，その子 $j \in T$ について，$\beta(j) \subseteq \beta(i)$ かつ $|\beta(i)| - |\beta(j)| \geq 2$ ならば，i と j の間に i'' $(\beta(i'') = \beta(i) \backslash \{v\}, v \in \beta(i) \backslash \beta(j))$ を加え，(i, i'', j) の順につなぐ.
   (c) ある $i \in T$ と，その子 $j \in T$ について，$\beta(i) \subseteq \beta(j)$ かつ $\beta(j) \backslash \beta(i)$ が強連結成分でないとき，i と j の間に i''' $(\beta(i''') = \beta(j) \backslash A$，$A$ は $\beta(j) \backslash \beta(i)$ の強連結成分) を加え，(i, i''', j) の順につなぐ.
   (d) ある i が k $(k \geq 3)$ 個以上の子を持つとき，i の直後に，高さが $\lfloor \log_2 k \rfloor$ の完全二分有向木 T' $(^\forall t' \in T'$ について $\beta(t') = \beta(i))$ を追加し，T' の各葉に対して 2 つずつ i の子を接続し，枝の向きは葉から子への向きとする．ただし，$\lfloor x \rfloor$ は x を超えない最大の整数値とする.
   (e) ある i が 2 つの子 j, k を持つとき，i と j，i と k の間にそれぞれ j', k' $(\beta(j') = \beta(k') = \beta(i))$ を追加し，それぞれ (i, j', j)，(i, k', k) の順につなぐ.

以上の操作によってできる DAG-TD は，width が高々w であり，また G の nice DAG-TD である．この構築は G の頂点数の多項式時間で行える．　□

6.3 計算量クラス

Theorem 11. 有向グラフ G が与えられたとき，G の *DAG-treewidth* を計算する問題は *NP* 困難である．

Theorem 11の証明は付録で示す．

6.4 co-DAG Tree Decomposition

DAG-TD T を FPT アルゴリズムに利用する場合，T の葉から動的計画法を開始すると FPT を構築しやすい．本節では，より FPT アルゴリズムの構築に適した DAG-TD である co-DAG Tree Decomposition を定義する．

Definition 30. 有向グラフ $G = (V, E)$ の co-DAG Tree Decomposition (co-DAG-TD) とは，反有向木 T と，T の各ノード t について $\beta(t) \subseteq V$ であるような β とのペア (T, β) である．(T, β) は以下の 3 つを満たす．なお，T の根の数を n_l とする．

1. $\sum_{t \in T} \beta(t) = V$
2. T の各葉を r_m $(m = 1, 2, \ldots, n_r)$ と表したとき，任意の枝 $(u, v) \in E$ について以下の少なくともいずれか一方が成り立つ．
   - ある j $(1 \leq j \leq n_r)$ があり，$u, v \in \beta(r_j)$
   - ある i $(i \in T)$ とその親 j があり，$u, v \in \beta(i)$, $u \notin \beta(j)$
3. 任意の $i, j, k \in T$ に対し，j が i から k までのパス上に存在するならば，$\beta(i) \cap \beta(k) \subseteq \beta(j)$．すなわち各 $v \in V$ について，v を含むバッグは T 上で非空な部分反有向木を誘導する．

以下で co-DAG-treewidth を定義する．

Definition 31 (co-DAG-treewidth)**.** ある co-DAG-TD (T, β) の width を $\max_{t \in T} |\beta(t)|$ と定義する．このとき，有向グラフ G の co-DAG-treewidth (co-DAG-treewidth) とは，G に対する全ての co-DAG-TD のうち，width が最小である co-DAG-TD の width である．

以降では，G の co-DAG-treewidth を $\mathsf{rtw}(G)$ と表す．

6.5 nice co-DAG-TD

本節では，nice DAG-TD と同様に nice co-DAG-TD (nice co-DAG-TD) を説明する．

Definition 32. $G = (V, E)$ を有向グラフとし，(T, β) を G の co-DAG-TD とする．(T, β) が以下の 2 つを満たすとき，(T, β) は G の nice co-DAG-TD (nice co-DAG-TD) であるという．

1. r_m $(m = 1, 2, \ldots, n_r)$ を T の各葉としたとき，全ての m に対して $\beta(r_m) = \varnothing$
2. $R = \{r_m | m = 1, 2, \ldots, n_r\}$ としたとき，各 $i \in T \backslash R$ に対し，$\beta(i)$ は以下のいずれかである．

   introduce i がただ 1 つの親 j をもち，ある強連結成分 S があり，$S \cap \beta(j) \neq \varnothing$ かつ $\beta(i) = \beta(j) \cup S$

   forget i がただ 1 つの親 j をもち，ある $v \in V$ があり，$\beta(i) = \beta(j) \backslash \{v\}$

   union i がちょうど 2 つの親 j, k をもち，$\beta(i) = \beta(j) = \beta(k)$

ある co-DAG-TD が与えられたとき，width が同じである G の nice co-DAG-TD を効率よく構築することができる．

Theorem 12. (T, β) が有向グラフ $G = (V, E)$ の *co-DAG-TD* であるとする．また (T, β) の *width* を w とする．このとき，*width* が w であるような G の *nice co-DAG-TD* を多項式時間で構築できる．

Proof. (T, β) に対し，以下の手順を行い nice co-DAG-PD を得る．

1. T の各葉 r_m に対し，親 r'_m $(\beta(r) = \varnothing)$ を追加する．
2. ある $i \in T$ がただ 1 つの親 $j \in T$ をもち，かつ $\beta(i) = \beta(j)$ ならば，T から j を除き，j のそれぞれの親から i への枝を加える．このような i, j がなくなるまでこの操作を繰り返す．
3. 以下の 5 つの状態がなくなるまで操作を繰り返す．
   (a) ある $i \in T$ と，その親 $j \in T$ について，$\beta(i) \nsubseteq \beta(j)$ かつ $\beta(j) \nsubseteq \beta(i)$ ならば，i と j の間に i' $(\beta(i') = \beta(i) \cap \beta(j))$ を加え，(j, i', i) の順につなぐ．
   (b) ある $i \in T$ と，その親 $j \in T$ について，$\beta(j) \subseteq \beta(i)$ かつ $|\beta(i)| - |\beta(j)| \geq 2$ ならば，i と j の間に i'' $(\beta(i'') = \beta(i) \backslash \{v\}, v \in \beta(i) \backslash \beta(j))$ を加え，(j, i'', i) の順につなぐ．
   (c) ある $i \in T$ と，その親 $j \in T$ について，$\beta(i) \subseteq \beta(j)$ かつ $\beta(j) \backslash \beta(i)$ が強連結成分でないとき，i と j の間に i''' $(\beta(i''') = \beta(j) \backslash A, A$ は $\beta(j) \backslash \beta(i)$ の強連結成分) を加え，(j, i''', i) の順につなぐ．
   (d) ある i が k $(k \geq 3)$ 個以上の親を持つとき，i の直前に，高さが $\lfloor \log_2 k \rfloor$

の完全二分反有向木 T' ($^\forall t' \in T'$ について $\beta(t') = \beta(i)$) を追加し，T' の各葉に対して 2 つずつ i の親を接続し，枝の向きは親から根への向きとする．ただし，$\lfloor x \rfloor$ は x を超えない最大の整数値とする．

(e) ある i が 2 つの親 j, k を持つとき，i と j，i と k の間にそれぞれ j', k' ($\beta(j') = \beta(k') = \beta(i)$) を追加し，それぞれ (j, j', i), (k, k', i) の順につなぐ．

以上の操作によってできる co-DAG-TD は，width が高々 w であり，また G の nice co-DAG-TD である．この構築は G の頂点数の多項式時間で行える． □

6.6 計算量クラス

Theorem 13. 有向グラフ G が与えられたとき，G の *co-DAG-treewidth* を計算する問題は *NP* 困難である．

Theorem 13の証明は付録で示す．

第7章　co-DAG-treewidthを用いたFPTアルゴリズム

7.1 Directed Dominating Set Problemの$O(2^w w^2 n)$時間アルゴリズム

以下では，頂点数nのDAG Gと幅がwであるGのco-DAG-TDが与えられたときに，GのDirected Dominating Set Problemを$O(2^w w^2 n)$で計算するアルゴリズムを示す．

Theorem 14. *DAG Gに対し，幅がwであるGの nice co-DAG-TDが与えられたとき，Gの DiDS problemを$O(2^w w^2 n)$で解くアルゴリズムが存在する．*

上記のアルゴリズムを構成するため，まず関数DSを定義する．

Definition 33 (DS)**.** DAG $G = (V, E)$のnice co-DAG-TDを(T, β)とし，その幅をwとする．ある$i \in T$に対し，頂点集合$A_i, B_i \subseteq V$が$A_i \cup B_i = \beta(i), A_i \cap B_i = \varnothing$を満たすとする．$G_i$を頂点集合$\bigcup_k \beta(k)$ (kはiを根とするTの部分反有向木のノード)によって誘導されるGの部分グラフとする．関数DSを以下のように定める．

$$\mathsf{DS}(i, A_i, B_i) = (\min |S_i|, D) \tag{13}$$

ただし$S_i \subseteq V[G_i]$, $D \subseteq B_i$を満たし，Dが支配されればS_iはG_iのminimum-DiDSであり，かつ$A_i \subseteq S_i$, $B_i \cap S_i = \varnothing$を満たすとする．

以下でDSの計算式を与える．各$\beta(i)$がintroduce, forget, unionのいずれかで場合分けをして計算する．

- iがただ一つの親jをもち，かつ$\beta(i)$が$v \in V$をintroduceしているとき

$$\mathsf{DS}(i, A_i, B_i) = \begin{cases} (\mathsf{DS}(j, A_i \backslash \{v\}, B_i)[0] + 1, \mathsf{DS}(j, A_i \backslash \{v\}, B_i)[1] \backslash \mathsf{suc}(v)) & (v \in A_i) \\ (\mathsf{DS}(j, A_i, B_i \backslash \{v\})[0], \mathsf{DS}(j, A_i, B_i \backslash \{v\})[1] \cup \mathsf{suc}(v)) & (v \in B_i) \end{cases}$$

- iがただ一つの親jをもち，かつX_iが$v \in V$をforgetしているとき

$$\mathsf{DS}(i, A_i, B_i) = \begin{cases} (\mathsf{DS}(j, A_i \cup \{v\}, B_i)[0], \mathsf{DS}(j, A_i \cup \{v\}, B_i)[1]) & (v \in \mathsf{DS}(j, A_i, B_i \cup \{v\})[1]) \\ (\mathsf{DS}(j, A_i, B_i \cup \{v\})[0], \mathsf{DS}(j, A_i, B_i \cup \{v\})[1]) & (otherwise) \end{cases}$$

• i が二つの親 j, k をもつとき

$$\mathsf{DS}(i, A_i, B_i) = (\mathsf{DS}(j, A_i, B_i)[0] + \mathsf{DS}(k, A_i, B_i)[0] - |A_i|, \mathsf{DS}(j, A_i, B_i)[1] \cap \mathsf{DS}(k, A_i, B_i)[1])$$

DS を用いて，DAG G の nice co-DAG-TD (T, β) が与えられたときに，G の mDiDS のサイズを出力するアルゴリズム Compute を示す．

Compute(T, β)

1. First Step: T の各根 r に対し，$\mathsf{DS}(r, \varnothing, \varnothing) = (0, \varnothing)$ とする．
2. Exection Step: 各バッグ $\beta(i)$ に対し，A_i, B_i 全ての組合せについて各根から順に $\mathsf{DS}(i, A_i, B_i)$ を計算する．
3. Final Step: i が T のただ一つの葉 l となったとき，$\mathsf{DS}(l, \varnothing, \varnothing)[0]$ を出力する．

Lemma 20. Compute は G の *mDiDS* のサイズを返す．

20の証明は付録で示す．

最後に Compute の計算量を示す．

Lemma 21. *DAG* G の頂点数が n であるとする．このとき，幅が w である G の *co-DAG-TD* (T, β) が与えられたとき，Compute(T, β) は $O(2^w w^2 n)$ の計算時間で結果を出力する．

Proof. 各バッグ $\beta(i)$ に対し，$|\beta(i)| \leq w + 1$ に注意すると，A, B の組合せは高々2^{w+1} 通り存在する．また (T, β) のバッグ数は高々$O(wn)$ である．さらに introduce, forget, union のそれぞれでの計算と条件の判定は高々$O(w)$ 時間を要する．以上より，Compute の計算量は $O(2^w w^2 n)$ である． □

第8章　結論

本章では本研究のまとめと今後の課題を示す．本研究では次の3つを行った．

まず1つ目にDAGパス幅をパラメータとして有向支配集合問題，最大葉分岐問題，k-有向点素パス問題，有向シュタイナー木問題を解くアルゴリズムを構築した．前提として入力グラフのパス分解がすでに与えられているとし，バッグがintroduceかforgetかで場合分けを行っている．これにより幅がwであるDAGパス分解を入力したときにそれぞれ$O(2^w wn)$, $O(2^w wn + n^2)$, $O((k+1)^w(w^2+k)n + n^2)$, $O(2^w(k+w)n + n^2)$で計算できることを示した．

2つ目にDAGパス分解を求める近似アルゴリズム及びパラメータ化アルゴリズムを構築した．近似アルゴリズムでは，DAG上のセパレータを利用した分割統治法を用いることで$O(\log^2 n)$-近似のアルゴリズムを構築した．またDAG上でのDAGパス分解の構築が*one-shot Black Pebbling*問題と等価であることを示し，近似比がより小さい$O(\log^{2/3} n)$-近似アルゴリズムの存在も示した．さらに入力DAGの最大出次数をd，根数をlとしたとき，入力整数kに対して幅が高々$O(ld^k)$のDAGパス分解を出力するか，DAGパス幅がkより大きいことの証拠を示すアルゴリズムを構築した．このアルゴリズムは完全有向d分木の埋め込みを利用して設計したもので，出次数が制限されたDAGに対して効率的にDAGパス分解を構築できることを初めて示した．

3つ目にDAGパス幅を拡張する概念としてDAG木幅を提案した．DAG木幅は有向グラフがどれだけ有向木に近い構造を持つかを表すパラメータであり，DAGパス幅より大きくならず，同様の手法でアルゴリズムを構築できる点が利点である．本論文ではDAG木幅を用いて有向支配集合問題を解く$O(2^w wn)$時間アルゴリズムを構築した．これによりDAGパス幅を用いたアルゴリズムと比較して計算速度の向上を実現した．

最後に今後の課題として2点挙げる．1つ目は$O(ld^k)$の幅を持つDAGパス分解を出力するアルゴリズムにおいて幅の値を小さくすることである．特に最大出次数dは頂点数nまで増大する可能性があるため，これを定数にまで抑える方法を検討したい．2つ目はDAGパス分解アルゴリズムを有向グラフ全体に拡張することである．本研究でのアルゴリズムはDAGに限定されているが，DAGパス幅自体は一般の有向グラフに対して定義されているため，幅の小さなDAGパス分解を一般の有向グラフに対しても適用できるようにする余地がある．

謝辞

最後に，本研究にあたって様々な助力をしていただいた指導教員である川原純准教授及び湊真一教授や岩政勇仁助教など湊研究室の方々に深謝いたします．また，私生活の面で 20 年以上にもわたって支えていただいた両親に感謝します．

参考文献

[1] Robertson, N. and Seymour, P.: Graph minors. I. Excluding a forest, *Journal of Combinatorial Theory, Series B*, Vol. 35, no.1, pp. 39–61 (1983).

[2] Robertson, N. and Seymour, P.: Graph minors. III. Planar tree-width, *Journal of Combinatorial Theory, Series B*, Vol. 36, no.1, pp. 49–64 (1984).

[3] Arnborg, S., Corneil, D. and Proskurowski, A.: Complexity of finding embeddings in a k-tree, *SIAM Journal on Algebraic Discrete Methods*, Vol. 8, no.2, pp. 277–284 (1987).

[4] Arnborg, S. and Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial k-trees, *Discrete Applied Mathematics*, Vol. 23, no.1, pp. 11–24 (1989).

[5] Amir, E.: Approximation Algorithms for Treewidth, *Algorithmica*, Vol. 56, pp. 448–479 (2010).

[6] Korhonen, T.: A single-exponential time 2-approximation algorithm for treewidth, *SIAM Journal on Computing*, pp. FOCS21–174 (2023).

[7] Groenland, C., Joret, G., Nadara, W. and Walczak, B.: Approximating pathwidth for graphs of small treewidth, *ACM transactions on algorithms*, Vol. 19, pp. 1–19 (2023).

[8] Cattell, K., Dinneen, M. J. and Fellows, M. R.: A simple linear-time algorithm for finding path-decompositions of small width, *Information processing letters*, Vol. 57, pp. 197–203 (1996).

[9] Reed, B.: Tree width and tangles: a new connectivity measure and some applications, *Surveys in Combinatorics*, pp. 87–162 (1997).

[10] Johnson, T., Robertson, N., Seymour, P. and Thomas, R.: Directed tree-width, *Journal of Combinatorial Theory, Series B*, Vol. 82, no.1, pp. 138–154 (2001).

[11] Berwanger, D., Dawar, A., P. Hunter, S. K. and Obdržálek, J.: The DAG-width of directed graphs, *Journal of Combinatorial Theory, Series B*, Vol. 102, no.4, pp. 900–923 (2012).

[12] Kasahara, S., Kawahara, J., Minato, S. and Mori, J.: DAG-pathwidth: Graph algorithmic analyses of DAG-type blockchain networks, *IEICE*

Transactions on Information and Systems, Vol. E106-D, No.3 (2023).

[13] Kirousis, L. and Papadimitriou, C.: Searching and pebbling, *Theoretical Computer Science*, Vol. 47, pp. 205–218 (1986).

[14] Gilbert, J. R., Lengauer, T. and Tarjan, R. E.: The pebbling problem is complete in polynomial space, *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pp. 237–248 (1979).

[15] Dziembowski, S., Faust, S., Kolmogorov, V. and K.Pietrzak: Proofs of Space, *Advances in Cryptology - CRYPTO 2015*, pp. 585–605 (2015).

[16] Sethi, R.: Complete Register Allocation Problems, *SIAM Journal on Computing*, Vol. 4, pp. 226–248 (1975).

[17] Ganian, R., Hliněnỳ, P., Kneis, J., Langer, A., Obdržálek, J. and Rossmanith, P.: Digraph width measures in parameterized algorithmics, *Discrete applied mathematics*, Vol. 168, pp. 88–107 (2014).

[18] Fomin, F. V., Grandoni, F., Kratsch, D., Lokshtanov, D. and Saurabh, S.: Computing optimal Steiner trees in polynomial space, *Algorithmica*, Vol. 65, pp. 584–604 (2013).

[19] Ravi, R., Agrawal, A. and Klein, P.: Ordering problems approximated: single-processor scheduling and interval graph completion, *Automata, Languages and Programming: 18th International Colloquium Madrid, Spain, July 8–12, 1991 Proceedings 18*, pp. 751–762 (1991).

[20] Austrin, P., Pitassi, T. and Wu, Y.: Inapproximability of treewidth, one-shot pebbling, and related layout problems, *International Workshop on Approximation Algorithms for Combinatorial Optimization*, Vol. 65, pp. 13–24 (2012).

[21] Even, S., Itai, A. and Shamir, A.: On the Complexity of Timetable and Multicommodity Flow Problems, *SIAM Journal on Computing*, Vol. 5, pp. 691–703 (1976).

[22] Belmonte, R., Hanaka, T., Katsikarelis, I., Kim, E. J. and Lampis, M.: New Results on Directed Edge Dominating Set, *CoRR*, Vol. abs/1902.04919 (2019).

[23] Hanaka, T., Nishimura, N. and Ono, H.: On directed covering and domination problems, *Discrete Applied Mathematics*, Vol. 259, pp. 76–99 (2019).

付録

A.1 4の証明

Proof. 以下では Compute の Preprocesing を行ったあとの場合を考える．すなわちグラフ G，nice DAG-PD は根 r から到達可能な頂点のみが現れるものとする．また Compute の First Step より，G が根 r のみからなる場合は 1 を出力するため MaxLOB の解となっている．以下では G に根 r 以外の頂点も含まれる場合を考える．Lemma 4の証明は，各 i に対し $\mathsf{LOB}(i, A_i, B_i)$ が LOB の定義 2を満たすことを示せば十分．これを i に関する数学的帰納法で示す．以下では便宜上，木を成す頂点のうち葉以外の頂点を茎と呼ぶ．

$i = 1$ のとき，Compute の First Step より $\mathsf{LOB}(1, \{r\}, \varnothing) = -\infty, \mathsf{LOB}(1, \varnothing, \{r\}) = 0$ である．根 r は葉となり得ないことに注意すると明らかに定義 2を満たす．

$i = k$ のとき，$\mathsf{LOB}(i, A_i, B_i)$ が定義 2を満たすと仮定する．以下で X_{k+1} が $v \in V$ を introduce するか forget するかで場合分けを行う．

- X_{k+1} が $v \in V$ を introduce する場合

  $v \in A_{k+1}$ の場合，DAG-PD のルール 2 より $\mathsf{pred}(v) \subseteq (A_{k+1} \cup B_{k+1})$ である．B_{k+1} は有向全域木の茎を表すことに注意すると，ある茎 $u \in \mathsf{pred}(v) \cap B_{k+1}$ が存在する場合に限り v を葉とすることができる．仮定より G_k において $\mathsf{LOB}(k, A_{k+1} \backslash \{v\}, B_{k+1})$ は，$A_{k+1} \backslash \{v\}$ を葉集合，B_{k+1} を茎集合とする葉数最大の有向全域木であり，これに v を葉として加えることで葉数を 1 だけ大きくできる．したがって $\mathsf{LOB}(k+1, A_{k+1} \backslash \{v\}, B_{k+1}) + 1$ は A_{k+1} を葉集合，B_{k+1} を茎集合とする有向全域木の葉数の最大値を表す．逆にこのような茎 u が存在しなければ v は葉とならないため，値を $-\infty$ とすることで解にならないことを表す．以上より $v \in A_{k+1}$ の場合は定義 2を満たす．

  $b \in B_{k+1}$ の場合も同様の議論により定義 2を満たす．ただし v は茎であるため LOB の値は大きくならないことに注意する．

- X_{k+1} が $v \in V$ を forget する場合

  葉数がそれぞれ $\mathsf{LOB}(k, A_{K+1} \cup \{v\}, B_{k+1}), \mathsf{LOB}(k, A_{k+1}, B_{k+1} \cup \{v\})$ である有向全域木を T_k^A, T_k^B と表す．まず $\mathsf{LOB}(k, A_{K+1} \cup \{v\}, B_{k+1}) \geq \mathsf{LOB}(k, A_{k+1}, B_{k+1} \cup \{v\})$ の場合，v が葉であるような T_k^A が実際に存在することを示す．このとき $\mathsf{LOB}(k, A_{K+1} \cup \{v\}, B_{k+1}) \neq -\infty$ であるため，v を introduce したバッグを X_j とすると，$i = j$ における LOB の値も $-\infty$ でない．

したがって式3の第一式の条件より，$i = j$ において v が葉であるような有向全域木の存在が保証される．X_m $(j < m < k+1)$ において v は introduce も forget もされないことに注意すると，T_k^A は実際に v を葉として含むことが示される．次に $\mathsf{LOB}(k, A_{K+1} \cup \{v\}, B_{k+1}) < \mathsf{LOB}(k, A_{k+1}, B_{k+1} \cup \{v\})$ の場合，v が茎であるような T_k^B が実際に存在することを示す．上記と同様の議論により，T_k^B には v の先行頂点が必ず存在する．また T_k^B において v と共通の子をもつ頂点も存在しない．なぜならば，もしそのような頂点 v' (ある $w \in V[T_k^B]$ があり，$\{(v, w),\ (v', w)\} \subseteq E[T_k^B]$) が存在したとすると，$T_k^B$ において v に w 以外の子が存在するならば，新たに $T_k^B = (V[T_k^B], E[T_k^B] \backslash \{(v, w)\})$ とすることで v と v' が共通の子を持たないようにすることができる．この操作を繰り返すことで v と共通の子をもつ頂点が存在しないような有向全域木を構築できる．T_k^B において v に w 以外の子が存在しないならば，$E[T_k^B]$ から (v, w) を除くことで v を葉とする有向全域木 T_k^A が構築できる．このとき明らかに $|\mathsf{Leaf}(T_k^A)| = |\mathsf{Leaf}(T_k^B)| + 1$ であるが，これは $\mathsf{LOB}(k, A_{K+1} \cup \{v\}, B_{k+1}) < \mathsf{LOB}(k, A_{k+1}, B_{k+1} \cup \{v\})$ と矛盾する．したがって T_k^B において v と共通の子をもつ頂点は存在せず，v が実際に茎であるような T_k^B が存在することが示される．

ここで $G_{k+1} = G_k$ より，G_{k+1} の有向全域木の最大葉数は G_k のそれと等しい．したがって v を葉に含むような G_k の有向全域木の最大葉数と，v を茎に含むような G_k の有向全域木の最大葉数のうち，より大きい方を G_{k+1} の有向全域木の最大葉数とすることができる．仮定より，それぞれ $\mathsf{LOB}(i, A_{i+1} \cup \{v\}, B_{i+1}), \mathsf{LOB}(i, A_{i+1}, B_{i+1} \cup \{v\})$ と表すことができるため，$\max\{\mathsf{LOB}(i, A_{i+1} \cup \{v\}, B_{i+1}), \mathsf{LOB}(i, A_{i+1}, B_{i+1} \cup \{v\})\}$ は G_{k+1} の有向全域木の最大葉数と等しい．したがって定義2が成立．

以上より，$i = k+1$ でも定義2が成立．数学的帰納法により Lemma 4が証明された． □

A.2 Theorem 11の証明

Theorem 11を証明するため以下の Lemma を示す．

Lemma 22. 有向グラフ $G = (V, E)$ の強連結成分 A があり，A から $V \backslash A$ への枝が存在しないとき，A を G の *sink* と呼ぶ．このとき，G が *sink* をただ *1* つしか持たないならば，$\mathsf{tw}(G) = \mathsf{pw}(G)$ が成り立つ．

Proof. DAG-PD, DAG-TD の定義より，width が $\mathsf{pw}(G)$ である G の DAG-PD は G の DAG-TD でもある．よって $\mathsf{tw}(G) \leq \mathsf{pw}(G)$ が成り立つ．一方，width が $\mathsf{tw}(G)$ である G の nice DAG-TD を (T, γ) とし，sink$A$ を含む T のバッグのうち，根 $r \in T$ からの距離が最も近いものを $\gamma(t_i)$ $(t_i \in T)$ とする．T 上で r から t_i までのパスを $P \subseteq T$ としたとき，(P, γ) が G の DAG-PD となっていることを示す．まず，P について $\sum_{p \in P} \gamma(p) = V$ が成り立つ．なぜならば，T 上で P に含まれない P' があり，かつ任意の $p \in P$ に対し，P' 上で，ある $v \notin \gamma(p)$ が introduce されたとすると，頂点集合 $\sum_{p' \in P'} \gamma(p')$ が誘導する G の部分有向グラフはある sinkA' をもつが，$p_A \in P$ $(A \subseteq \gamma(p_A))$ と $p_{A'} \in P'$ $(A \subseteq \gamma(p_{A'}))$ は非連結であるから，DAG-TD の定義より，$A \neq A'$ である．これは G が sink をただ 1 つしかもたないことと矛盾する．したがって $\sum_{p \in P} \gamma(p) = V$ が成り立つことがいえ，(P, γ) は DAG-PD のルール 1 を満たす．また DAG-TD のルール 2 と，P がパスであることから (P, γ) は DAG-PD のルール 2 を満たす．さらに P はパスであるからルール 3 も満たす．よって (P, γ) は G の DAG-PD である．したがって $\mathsf{pw}(G) \leq \mathsf{tw}(G)$ がいえる．以上より，$\mathsf{tw}(G) = \mathsf{pw}(G)$ が示される． □

上の Lemma を用い，[3] の手法を参考にして Theorem 11を示す.

proof of Theorem 11. ある有向グラフ $G = (V, E)$ が与えられたとき，以下の手順で有向グラフ $G' = (V', E')$ $(|V'| = 2|V| + 1, |E'| = \frac{1}{2}(|V|^2 + 3|V|) + 2|E|)$ を構成する．

- $V = \{v_i | i = 1, 2, \ldots, n\}$ としたとき，$V' = \{v_i, v'_i | i = 1, 2, \ldots, n\} \cup \{s\}$ とする．
- (v_i, v_j) $(v_i, v_u \in V', 1 \leq i < j \leq n)$ の全ての組合せに対し，$(v_i, v_j) \in E'$ とする．
- 各 i $(i = 1, 2, \ldots, n)$ に対し，(v_i, v'_i), $(v'_i, s) \in E'$ とする．
- $v_p, v_q \in V$ について $(v_p, v_q) \in E$ であるならば，(v_p, v'_q), (v_q, v'_p), $(v'_p, v'_q) \in E'$ とする．

以上の操作によってできる有向グラフを G' とする．G' の構成例を Figure?? に示す．G' は G のサイズの多項式時間で構成できる．また G' の頂点集合 A, B を，$A = \{v_i | i = 1, 2, \ldots, n\}, B = \{v'_i | i = 1, 2, \ldots, n\}$ と定める．このとき $V' = A \cup B \cup \{s\}$ とできる．

次に，width が $\mathsf{pw}(G)$ である G の DAG-PD を (P, β) $(P = p_1, p_2, \ldots, p_r)$ とする．各 m に対し，G' の頂点集合 $\gamma(p_m)$ を以下のように構成する．

- 各 (p_m) に対し，$v \in \bigcup_{j=i}^{r} \beta(p_j)$ ならば，v を $\gamma(p_m)$ に加える．
- 各 (p_m) に対し，$v \in \bigcup_{j=1}^{i} \beta(p_j)$ ならば，v' を $\gamma(p_m)$ に加える．

さらに $\gamma(p_{r+1})$ を下記のように定める．

- $\gamma(p_{r+1}) = B \cup \{s\}$

以上の操作によってできる (P', γ) $(P' = p_1, p_2, \ldots, p_{r+1})$ が，G' の DAG-PD になっていることを示す．Figure??の G, G' に対する DAG-PD (P, β), (P', γ) の構成例を Figure??で示す．γ の構成法に注意すると，$\gamma(p_1)$ は A の頂点をすべて含む．また $\gamma(p_{r+1})$ は $B \cup \{s\}$ を含む．よって $\sum_{m=1}^{r+1} \gamma(p_m) = V'$ が成り立つため，(P', γ) は DAG-PD のルール 1 を満たす．また γ の構成法に注意すると，任意の $v \in V'$ に対して，v を含む全てのバッグが $(p_1, P_2, \ldots, p_r)$ 上で非連結になることはなく，また $\gamma(p_r)$ は B の頂点をすべて含み，s は $\gamma(p_{r+1})$ にのみ含まれるため，任意の $v \in V'$ に対して，v を含む全てのバッグが $(p_1, P_2, \ldots, p_r + 1)$ 上で非連結になることはない．よって (P', γ) は DAG-PD のルール 3 を満たす．さらに，(P', γ) が G' の全ての枝に対して DAG-PD のルール 2 を満たすことを示す．G' 上の各枝 $e \in E'$ を次の 5 つに場合分けして考える．なお，v, v' はそれぞれ A, B に含まれることに注意する．

- $e = (v_i, v_j)$ $(1 \leq i < j \leq n)$ の場合：
  $A \subseteq \gamma(p_1)$ より，$v_i, v_j \in \gamma(p_1)$
- $e = (v'_i, s)$ $(1 \leq i \leq n)$ の場合：
  $B \cup \{s\} = \gamma(p_{r+1}), s \notin \gamma(p_r)$ より，$v'_i, s \in \gamma(p_{r+1}), s \notin \gamma(p_r)$
- $e = (v_i, v'_i)$ $(1 \leq i \leq n)$ の場合：
  γ の構成より，$v_i \in \beta(p_1)$ ならば $v_i, v'_i \in \gamma(p_1)$．$v_i \notin \beta(p_1)$ ならば，(P, β) で v が初めて現れたバッグを $\beta(p_m)$ とすると，$v_i, v'_i \in \gamma(p_m)$ であり，v'_i は $\gamma(p_1)$ で初めて現れている．よって $v_i, v'_i \in \gamma(p_m), v'_i \notin \gamma(p_{m-1})$
- $e = (v_i, v'_j)$ $((v_i, v_j) \in E \in G)$ の場合：
  γ の構成より，$v_i, v_j \in \beta(p_1)$ ならば $v_i, v'_j \in \gamma(p_1)$．$v_i, v_j \notin \beta(p_1)$ ならば，(P, β) において，ある m があり，$v_i, v_j \in \beta(p_m), v_j \neq \beta(p_{m-1})$ とできる．こ

のとき，γ の構成より $v_i, v'_j \in \gamma(p_m), v'_j \notin \gamma(p_{m-1})$

- $e = (v_j, v'_i)$ $((v_i, v_j) \in E \in G)$ の場合：
  (P, β) において，v_i が初めて現れたバッグを $\beta(p_m)$ とする．このとき γ の構成より，v'_i は $\gamma(p_m)$ で初めて現れている．また (P, β) は G の DAG-PD であるから，ある m' があり，$v_i, v_j \in \beta(p'_m), v_j \neq \beta(p_{m'-1})$ とできる．γ の構成より，$v_j \in \gamma(p_l)$ $(p_l = 1, 2, \ldots, m')$ がいえ，さらに $m \leq m'$ に注意すると $v_j \in \gamma(p_m)$ がいえる．したがって $v_j, v'_i \in \gamma(p_m), v_j \notin \gamma(p_{m-1})$

よって，(P', γ) は DAG-PD のルール 2 を満たす．以上より (P', γ) が G' の DAG-PD であることが示される．また，各 $m = 1, 2, \ldots, r$ に対し，$v_i \in \beta(p_m)$ ならば $v_i, v'_i \in \gamma(p_m)$ であり，かつ $v_i \notin \beta(p_m)$ ならば，v_i, v'_i のどちらか一方が $\gamma(p_m)$ に含まれるため，$|\gamma(p_{r+1})| = n + 1$ に注意すると，(P', γ) の width は $((P, \beta)$ の $width) + n$，すなわち $\mathsf{pw}(G) + n$ であることがいえる．

次に，(P', γ) が G' の DAG-PD のうち最小の width をもつ DAG-PD であることを示す．(P', γ) よりも width の小さい DAG-PD (Q, δ) $(Q = (q_1, q_2, \ldots, q_{r'}))$ が存在すると仮定し，矛盾を導くことでこれを示す．まず，以下の手順によって (Q, δ) と同じ幅をもつ G' の nice DAG-PD を構成する．

1. (Q, δ) のバッグのうち，初めて A のすべての頂点が含まれたバッグを $\delta(q_i)$ $(q_i \in Q)$ とする (A の任意の 2 頂点間には枝があるため，このような q_i が存在することに注意する)．ある $v' \in B$ に対し，$v' \in \delta(q_j)$ なる q_j $(j < i)$ が存在する場合，このような q_j を Q からすべて取り除く．$\delta(q_j) \subseteq \delta(q_i)$ であったことに注意すると，q_j の削除後も width が同じ G' の DAG-PD になっており，こうしてできる DAG-PD を (Q', δ') とする．
2. Q' の左端に q_0 $(\delta'(q_0) = A)$ を追加し，(Q'', δ'') とする．
3. (Q'', δ'') のバッグのうち，初めて $B \cup \{s\}$ のすべての頂点が含まれたバッグを $\delta''(q_k)$ $(q_k \in Q'')$ とする (B のすべての頂点から S に向かう枝があるため，このような q_k が存在することに注意する)．$v \in \delta''(q_k)$ なる $v \in A$ が存在する場合，Q'' から $\{q_j | j \geq k\}$ をすべて取り除き，q_{k-1} の直後に，順に q'_k, q'_{k+1}, q'_{k+2} $(\delta''(q'_k) = \delta(q_k) \backslash \{s\}, \delta''(q'_{k+1}) = B, \delta''(q'_{k+2}) = B \cup \{s\})$ を追加する．この操作を行っても DAG-PD のルールは満たされたままであり，かつ width は操作前より大きくならないことに注意する．こうしてできる DAG-PD を (Q''', δ''') とする．

4. (Q''', δ''') を width が同じ nice DAG-PD に変換する．便宜上，この nice DAG-PD を (Q, δ) $(Q = (q_1, q_2, \ldots, q_{r'})$ と表記する．

以上の操作によって構成された nice DAG-PD (Q, δ) は，はじめ A の頂点を順に introduce し，ある q_i で A の全ての頂点の introduce が終わった後，B の頂点の introduce を行いつつ，A の頂点の foget も行っていく．B のすべての頂点の introduce が終わった後，ある q_j で B のみを含むバッグがあり，直後に $B \cup \{s\}$ からなるバッグが続く．Figure**??**で構成後の例を示す．このとき，$\delta(q_k)$ $(q_k \in Q, i < k < j)$ では，各 v_m, v'_m $(m = 1, 2, \ldots, n)$ について，v_m, v'_m の少なくともどちらか一方は $\delta(q_k)$ に含まれる．なぜならば，G' には枝 (v_m, v'_m) が存在し，v_m, v'_m を同時に含む $\delta(q_{i_m})$ が存在すること，そして $\delta(q_i) = A, \delta(q_j) = B$ に注意すると，$v_m \in \delta(q_{i_L}), v'_m \in \delta(q_{i_R})$ $(q_{i_L}, q_{i_R} \in Q, i \leq i_L \leq i_m \leq i_R \leq j)$ がいえるからである．

ここで各 m $(m = 1, 2, \ldots, r')$ に対し，G の頂点集合 $\beta'(q_m)$ を $\beta'(q_m) = \{v_i | \{v_i, v'_i\} \subseteq \delta(q_m)\}$ と定める．このとき，(Q, β') が G の DAG-PD になっていることを示す．G' において任意の頂点対 (v, v') $(v \in A, v' \in B)$ が隣接していることに注意すると，(Q, δ) では (v, v') を同時に含むようなバッグが少なくとも 1 つ存在する．よって $\sum_{m=1}^{r'} \beta'(q'_r) = V$ が成り立つため，(Q, β') は DAG-PD のルール 1 を満たす．また G' のうち B によって誘導される部分グラフを G'_B とすると，$E(G'_B) = E(G)$ であるため，B の頂点で introduce される順序は，G' の強連結成分に対するトポロジカル順序になる．したがって (Q, β') も G' の強連結成分がトポロジカル順序で introduce されていくため，(Q, β') は DAG-PD のルール 2 を満たす．さらに，(Q, δ) は G' の DAG-PD であるから，(Q, δ) で A の頂点が一度 forget された後，再び introduce されることはない．よって (Q, δ) で v, v' を含むバッグが非連結になることはないため，(Q, β') でも v を含むバッグが非連結になることはない．したがって DAG-PD のルール 3 を満たす．

以上より，(Q, β') は G の DAG-PD になっている．このとき，(Q, β') の各バッグは，(Q, δ) の各バッグが (v_i, v'_i) を同時に含む場合のみ v_i を含んでいること，そして各 $\delta(q_m)$ には v_i, v'_i の少なくともどちらか一方が含まれることに注意すると，(Q, δ) の width k に対し，(Q, β') の width w_b は高々 $k - n$ である．ここで仮定より $k < \mathsf{pw}(G) + n$ であるため $w_b < \mathsf{pw}(G)$ とできるが，これは $\mathsf{pw}(G)$ が G の DAG-pathwidth であることと矛盾する．したがって仮定が誤りであり，(P', γ) が G' の DAG-PD のうち最小の width をもつ DAG-PD であることが示される．

以上より，$\mathsf{pw}(G') = \mathsf{pw}(G) + n$ が導かれる．G' が sink をただ 1 つもつことと Lemma 22 とより $\mathsf{tw}(G') = \mathsf{pw}(G')$ がいえるため，$\mathsf{pw}(G') = \mathsf{tw}(G') = \mathsf{pw}(G) + n$ が導かれる．

□

A.3 Theorem 13の証明

Theorem 13を証明するため，次で定義される co-DAG-pathwidth の計算が NP 困難であることを示す．

Definition 34. $G = (V, E)$ とする．G の co-DAG Path Decomposition (co-DAG-PD) とは，以下の 3 つの条件をみたすような V の各部分集合 $X_i (i = 1, 2, \ldots, s)$ の列 $X = (X_1, X_2, \ldots, X_s)$ である．

1. $X_1 \cup X_2 \cup \cdots \cup X_s = V$
2. 任意の有向枝 $(u, v) \in E$ について，以下のいずれかが成り立つ．
   - $u, v \in X_1$
   - ある i $(i \geq 2)$ があり，$u, v \in X_i$, $u \notin X_{i-1}$ が成り立つ．
3. 任意の i, j, k $(1 \leq i \leq j \leq k \leq s)$ について，$X_i \cap X_k \subseteq X_j$ が成り立つ．
   すなわち，各頂点 v について，v は X 上で非空な部分道を誘導する．

ある co-DAG-PD $P = (X_1, X_2, \ldots, X_s)$ の width を $\max_{1 \leq i \leq s} X_i$ と定義する．このとき，有向グラフ G の co-DAG-pathwidth とは，G に対する全ての co-DAG-PD のうち，width が最小である co-DAG-PD の width である．

以降では，G の co-DAG-pathwidth を $\mathsf{cpw}(G)$ と表す．

Lemma 23. 有向グラフ G，整数 k が与えられたとき，$\mathsf{cpw}(G) \leq k$ が成り立つかを判定する問題は *NP* 完全である．

Proof. G のある co-DAG-PD が与えられたとき，その width が k 以下かどうかを判定することは明らかに多項式時間で行えるため，この問題は NP に属する．また，G の全ての枝の向きを逆にした有向グラフを G' とする．G に対して width が $\mathsf{pw}(G)$ である DAG-PD P が与えられたとき，G' に対して最小の width をもつ co-DAG-PD P' は，DAG-PD と co-DAG-PD 定義より P と等しい．したがって P' の width は $\mathsf{pw}(G)$ であるため，$\mathsf{cpw}(G') = \mathsf{pw}(G)$ が成り立つ．以上より Lemma 23が示される． □

また，次の Lemma を示す．

Lemma 24. $G = (V, E)$ が *sink* をただ *1* つしか持たないならば，$\mathsf{ctw}(G) = \mathsf{cpw}(G)$ が成り立つ．

Proof. co-DAG-PD, co-DAG-TD の定義より，width が $\mathsf{cpw}(G)$ である G の co-DAG-PD は G の co-DAG-TD でもある．よって $\mathsf{ctw}(G) \leq \mathsf{cpw}(G)$ が成り立つ．一方，width が $\mathsf{ctw}(G)$ である G の nice co-DAG-TD を (T, γ) とし，G のただ一つの強連結成分である sink を $A \subseteq V$ とする．このとき nice co-DAG-TD の定義より，ある根 r とその子 r_c があり，$\gamma(r) = \varnothing, \gamma(r_c) = A$ が成り立つ．このとき T はただ一つの根 r をもつことがいえる．なぜならば，もし T が r 以外の根 r' $(\gamma(r') = \varnothing)$ をもつと仮定すると，r' の子はバッグに A を含まないため，G に A 以外の sink が存在することになり，矛盾するからである．したがって T は union のノードを持たず，任意の $t \in T$ は入次数と出次数が高々1 である．したがって co-DAG-PD と co-DAG-TD の定義より，(T, γ) は G の DAG-PD でもある．これより $\mathsf{cpw}(G) \leq \mathsf{ctw}(G)$ がいえる．以上より，$\mathsf{ctw}(G) = \mathsf{cpw}(G)$ が示される． □

最後に Theorem 13を示す．

Proof. ある有向グラフ $G = (V, E)$ が与えられたとき，定義11の証明と同様の手順で $|V'| = 2|V| + 1, |E'| = \frac{1}{2}(|V|^2 + 3|V|) + 2|E|$ である有向グラフ $G' = (V', E')$ を構成する．また G' の頂点集合 A, B を同様に $A = \{v_i | i = 1, 2, \ldots, n\}, B = \{v'_i | i = 1, 2, \ldots, n\}$ と定める．このとき $V' = A \cup B \cup \{s\}$ とできる．G' は G のサイズの多項式時間で構成できる．

次に，width が $\mathsf{cpw}(G)$ である G の co-DAG-PD を (P, β) $(P = p_1, p_2, \ldots, p_r)$ とする．各 m に対し，G' の頂点集合 $\gamma(p_m)$ を以下のように構成する．

- 各 (p_m) に対し，$v \in \bigcup_{j=i}^{r} \beta(p_j)$ ならば，v' を $\gamma(p_m)$ に加える．
- 各 (p_m) に対し，$v \in \bigcup_{j=1}^{i} \beta(p_j)$ ならば，v を $\gamma(p_m)$ に加える．

さらに $\gamma(p_0) = \gamma(p_1) \cup \{s\}$ $((p_0, p_1) \in E[T])$ と定める．

以上の操作によってできる (P', γ) $(P' = (p_0, p_1, p_2, \ldots, p_r))$ が G' の co-DAG-PD になっていることを示す．γ の構成法に注意すると，$A \subseteq \gamma(p_r), B \subseteq \gamma(p_1)$ がいえ，さらに $r \in \gamma(p_0)$ であることから，$\sum_{m=0}^{r} \gamma(p_m) = V'$ が成り立つ．よって (P', γ) は co-DAG-PD のルール 1 を満たす．また γ の構成法に注意すると，任意の $v \in V' \backslash \{r\}$ に対して，v を含む全てのバッグが $(p_1, p_2, \ldots, p_r)$ 上で非連結に

なることはなく，また $B \cup \{r\} \subseteq \gamma(p_0)$ より，任意の $v \in V'$ に対して，v を含む全てのバッグが $(p_0, p_1, p_2, \ldots, p_r)$ 上で非連結になることはない．よって (P', γ) は co-DAG-PD のルール 3 を満たす．さらに定義 11の証明と同様の議論により，(P', γ) が co-DAG-PD のルール 2 を満たすことがいえる．以上より (P', γ) は G' の co-DAG-PD である．このとき (P', γ) の幅は $\mathsf{cpw}(G) + n$ である．

さらに 11の証明と全く同様の議論により，(P', γ) が G' に対して width が最小の co-DAG-PD であることがいえる．ここで G' は sink を一つしか持たないことに注意すると，Lemma 24より $\mathsf{ctw}(G') = \mathsf{cpw}(G')$ がいえ，$\mathsf{ctw}(G) = \mathsf{cpw}(G) + n$ が成り立つ．以上より定義 13が示される． □

A.4 Lemma 20の証明

Proof. $i = l$ のとき，定義 13より DS は mDiDS のサイズを出力する．したがって各 $i \in T$ に対し，$\mathsf{DS}(i, A_i, B_i)$ が DS の定義 1を満たすことを示せば十分．これを i に関する数学的帰納法で示す．まず i が T の根のとき $\beta(i) = \varnothing$ であり，First Step より明らかに定義 13を満たす．次にある i での定義 13の成立を仮定する．以下で i の子 i' が introduce か forget か union かで場合分けを行う．

- i' が $v \in V$ を introduce する場合

  i' はただ一つの親 i をもつ．$v \in A_{i'}$ の場合，co-DAG-TD のルール 2 より v の後続頂点は i' の時点で全て introduce されており，v は $\mathsf{suc}(v)$ を支配する．また co-DAG-TD のルール 2 より v は i' の時点でどの頂点からも支配されていない．仮定より $D = \mathsf{DS}(i, A_{i'}, B_{i'})[1]$ が支配されれば G_i の mDiDS のサイズは $\mathsf{DS}(i, A_{i'}, B_{i'})[0]$ であるから，$D \backslash \mathsf{suc}(v)$ が支配されれば $G_{i'}$ の mDiDS のサイズは $\mathsf{DS}(i, A_{i'}, B_{i'})[0] + |v|$ となる．したがって i' でも定義 13を満たす．

  $v \in B_{i'}$ の場合，v は支配集合に含まれないので i' 以降に introduce される頂点によって支配される必要がある．仮定より $D = \mathsf{DS}(i, A_{i'}, B_{i'})[1]$ が支配されれば G_i の mDiDS のサイズは $\mathsf{DS}(i, A_{i'}, B_{i'})[0]$ であるから，$D \cup \{v\}$ が支配されれば $G_{i'}$ の mDiDS のサイズは $\mathsf{DS}(i, A_{i'}, B_{i'})[0]$ となる．したがって i' でも定義 13を満たす．

- i' が $v \in V$ を forget する場合

  i' はただ一つの親 i をもつ．$v \in \mathsf{DS}(i, A_{i'}, B_{i'} \cup \{v\})[1]$ の場合，v を支配する頂点は $A_{i'}$ に存在しないため，i の時点で v 自身が支配集合に含まれてい

る必要がある．逆にこのとき，仮定より $\mathsf{DS}(i, A_{i'} \cup \{v\}, B_{i'})[1]$ が支配されれば G_i の mDiDS のサイズは $\mathsf{DS}(i, A_{i'} \cup \{v\}, B_{i'})[0]$ であるため，i' でも定義 13を満たす．

$v \notin \mathsf{DS}(i, A_{i'}, B_{i'} \cup \{v\})[1]$ の場合，v を支配する頂点が $A_{i'}$ に少なくとも 1 つ存在する．したがって v は i の時点で $A_{i'}, B_{i'}$ のいずれに含まれていてもよく，また mDiDS のサイズが小さくなるのは v が $B_{i'}$ に含まれている場合である．仮定より $\mathsf{DS}(i, A_{i'}, B_{i'} \cup \{v\})[1]$ が支配されれば G_i の mDiDS のサイズは $\mathsf{DS}(i, A_{i'}, B_{i'} \cup \{v\})[0]$ であるため，i' でも定義 13を満たす．

- i' が二つの親 $i, \bar{i}$ をもつ union であるとき

  $\bar{i}$ においても定義13が成立していると仮定する．また $D_i = \mathsf{DS}(i, A_i, B_i)[1], D_{\bar{i}} = \mathsf{DS}(\bar{i}, A_{\bar{i}}, B_{\bar{i}})[1]$ とする．以下では $A_{i'} = A_i = A_{\bar{i}}, B_{i'} = B_i = B_{\bar{i}}$ であることに注意する．$D_i = D_{\bar{i}}$ の場合，仮定より D_i が支配されれば S_i は G_i の mDiDS であり，かつ D_i が支配されれば $S_{\bar{i}}$ は $G_{\bar{i}}$ の mDiDS である．このとき $S_i \cap S_{\bar{i}} = A_{i'}$ が成り立つ．なぜならば定義 13より $B_i \cap S_i = \varnothing, B_{\bar{i}} \cap S_{\bar{i}} = \varnothing$ であり，co-DAG-TD のルール 3 より任意の $s \in S_i \cap S_{\bar{i}}$ を含むバッグは T 上で $i', i, \bar{i}$ を含む連結な反有向木を形成しているからである．したがって $V[G_{i'}] = V[G_i] \cup V[G_{\bar{i}}]$ に注意すると，D_i が支配されれば $S_i \cup S_{\bar{i}}$ は明らかに $G_{i'}$ の mDiDS である．このとき mDiDS のサイズは $\mathsf{DS}(i, A_{i'}, B_{i'})[0] + \mathsf{DS}(\bar{i}, A_{i'}, B_{i'})[0] - |A_{i'}|$ であるため i' でも定義 13を満たす．

  $D_i \neq D_{\bar{i}}$ の場合，$D_{\bar{i}} \backslash D_i \neq \varnothing$ としても一般性を失わない．このときある頂点 $w \in D_{\bar{i}} \backslash D_i$ が存在する．定義 13より w は $B_{\bar{i}} = B_i = B_{i'}$ に含まれ，かつ w はある頂点 $w' \in S_i \subseteq V[G_i]$ によって支配されており，$S_{\bar{i}}$ には w を支配する頂点が存在しない．ここで $V[G_{i'}] = V[G_i] \cup V[G_{\bar{i}}]$ であるから，$S_{i'} = S_i \cup S_{\bar{i}}$ とすることで $S_{i'}$ には w を支配する w' が含まれる．したがって仮定に注意して $D_i \cap D_{\bar{i}}$ が支配されれば $S_i \cup S_{\bar{i}}$ は明らかに $G_{i'}$ の mDiDS である．上記と同様の議論により $|S_i \cup S_{\bar{i}}| = \mathsf{DS}(i, A_{i'}, B_{i'})[0] + \mathsf{DS}(\bar{i}, A_{i'}, B_{i'})[0] - |A_{i'}|$ であるため，i' においても定義 13を満たす．

以上より，i' でも定義 13が成立．数学的帰納法により Lemma 20が証明された．

□