修士論文

DAG パス幅に基づく種々のアルゴリズムの設計及び DAG 木幅への拡張

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 情報学専攻 通信情報システムコース

伊豆 真哉

2025 年 1 月 31 日

DAG パス幅に基づく種々のアルゴリズムの設計及び DAG 木幅への拡張

伊豆 真哉

内容梗概

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，既存の幅で表現することが難しい 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．

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 の 2 つの問題に対しては，頂点数 n の DAG と幅が w である DAG パス分解が与えられたときに共に $O(2^w wn)$ 時間で厳密解を与える 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 $O(2^w(k+w)n+n^2)$ 時間で厳密解を与える 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．DAG パス分解を構成する問題は One Shot Black Pebbling Problem と等価であることを示し，これによりさらに近似比を $O(\log^{3/2} n)$ まで小さくできることを示す．

DAG パス幅の計算は NP 困難であることが知られており，小さな DAG パス幅を求めるアルゴリズムは我々の知る限り知られていなかった．本研究では最大出次数，根数がそれぞれ d, l の DAG G と整数 k が与えられたとき，幅が高々 $O(ld^k)$ である G の DAG パス分解か，もしくは G の DAG パス幅が k よりも大きい証拠である DAG パス分解のいずれかを返すアルゴリズムを提案する．

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 $O(2^w(k+w)n+n^2)$ 時間の FPT アルゴリズムを提案する.

Designing various algorithms based on DAG-pathwidth and extension to DAG-treewidth

Shinya Izu

Abstract

abstruct

DAG パス幅に基づく種々のアルゴリズムの設計及び DAG 木幅への拡張

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，$O(2^w wn)$ 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．第 3 章の最後では Directed Edge Dominating Set Problem を用いて木分解と比較したときの DAG パス分解の利点を述べる．提案するアルゴリズムは DAG を入力としているが，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，根数がそれぞれ w, d, l である DAG が与えられたとき，$O(l \cdot d^w)$ の幅をもつ DAG パス分解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 分木の埋め込み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 $O(2^w w^2 n)$ 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．

次に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．

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

またDAGの根及び葉は以下のように定義される．

Definition 3. DAG $G = (V, E)$ に対し，ある頂点 $v \in V$ が G の根であるとは，v の入次数が0であることである．同様に $u \in V$ が G の葉であるとは，v の出次数が0であることである．

一般にDAGの根及び葉は複数存在する．

さらに特殊なDAGとして有向木を定義する．

Definition 4. DAG $G = (V, E)$ が以下をすべて満たすとき，G は有向木である．

1. G はただ一つの根 r をもつ．
2. G の枝の向きを無視した基礎グラフが木である．
3. 任意の頂点 $v \in V$ に対し，r から v への有向パスがただ1つ存在する．

上記で定義される有向木に対し，全ての枝の向きを逆にしたものを反有向木と呼ぶ．ただし反有向木において出次数が0の頂点を根，入次数が0の頂点を葉と表現する．

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 5 (パス分解)**.** [1] $G = (V, E)$ を無向グラフとする．G のパス分解

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 6 (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 7 (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 8 (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 9 (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 10 (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 11 (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．

Lemma 1の証明は付録で示す．

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 12 (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 パス分解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 13 (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 14 (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 nice DAG パス分解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 15 (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* パス分解が与えられたとき，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．DS は各 i においてバッグ X_i を共通部分をもたない頂点集合 $A_i, B_i \subseteq X_i$ に分ける．その後 $X_1 \cup X_2 \cup \cdots \cup X_i$ によって誘導される部分グラフ G_i の有向支配集合のうち，

A_i を全て含み，B_i を全て含まず，かつサイズが最小の有向支配集合を求める．これを全ての A_i, B_i の組合せについて計算することで，G_i での最小支配集合を得る．全ての i について上記の計算を行うことで最終的に入力グラフ G の最小支配集合を得る．

Definition 16 (DS)**.** DAG $G = (V, E)$ の nice DAG パス分解を $P = (X_1, X_2, \ldots, X_s)$ 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 パス分解 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 パス分解のルール 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* パス分解 P が与えられたとき，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 $O(2^w wn)$ 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 nice DAG パス分解, 根 r が与えられたとき，G の Max Leaf Outbranching Problem を $O(2^w wn)$ 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 17 (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，r を根とする 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.

今回は入力グラフを DAG とし，入次数が 0 の頂点がただ 1 つ存在し，その頂点が根 r であるとする．このとき G には葉数が最大の有向全域木が必ず存在する．これは次の補題を示せば十分である.

Lemma 4. 連結な *DAG* G について，G が入次数 *0* の根をただ一つもつことと，G に有向全域木が存在することは必要十分である.

Proof. G に入次数 0 の頂点が 2 つ以上存在する場合，明らかに有向全域木が存在しない．逆に入次数 0 のただ 1 つの頂点 r が存在する場合，G の各頂点を r を先頭とするトポロジカル順序に並べる．これを $r, v_1, v_2, \ldots, v_{n-1}$ とする．G は連結な DAG であることに注意すると，r 以外の各頂点 v_i $(i = 1, 2, \ldots, n-1)$ は $r, v_1, v_2, \ldots, v_{i-1}$ に少なくとも 1 つの親を持つ．これを全ての v_i に対して考

えることにより，各 v_i は r からの到達可能なパス P_i が存在する．全ての P_i の和を考えたときに有向木であれば，それは有向全域木である．有向木でなく閉路が存在する場合，ある 2 つのパス P_i, P_j $(i \neq j)$ があり，それぞれのパスの内部に P_i', P_j' (P_i', P_j' は始点と終点が一致する点素なパス) を含むとすることができる．このときパス P_i の P_i' を P_j' に変えることで P_i', P_j' による閉路を削除できる．この操作を全ての閉路に対して行うことにより，全てのパス P_i の和を有向木に変換できる．このとき有向木は有向全域木である． □

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 2. *DAG G に対し，幅が w である G の nice DAG パス分解が与えられたとき，G の MaxLOB 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 LOB を定義する．LOB は各 i においてバッグ X_i を共通部分をもたない頂点集合 $A_i, B_i \subseteq X_i$ に分ける．その後 $X_1 \cup X_2 \cup \cdots \cup X_i$ によって誘導される部分グラフ G_i の有向全域木のうち，A_i が葉であり，B_i が葉でなく，かつ葉数が最大のものを求める．これを全ての A_i, B_i の組合せについて計算することで，G_i での最大葉数の有向全域木を得る．全ての i について上記の計算を行うことで最終的に入力グラフ G の最大葉数の有向全域木を得る．

Definition 18 (LOB)**.** DAG $G = (V, E)$ の nice DAG パス分解を $P = (X_1, X_2, \ldots, X_s)$ 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 パス分解 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. 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．
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 $\mathsf{LOB}(i, A_i, B_i)$ を計算する．
3. Final Step: $i = s$ ならば，$\mathsf{LOB}(s, \varnothing, \varnothing)$ を出力する．

Lemma 5. Compute は G の $MaxLOB$ の解を出力する．

Lemma 5の証明は付録で示す．

最後に Compute の計算量を示す．

Lemma 6. 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 パス分解 P が与えられたとき，Compute(P) は $O(2^w wn)$ の計算時間で結果を出力する．

Proof. 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 $\mathsf{pred}(v)$ の計算量が高々$O(w)$ であることに注意すると，$\mathsf{LOB}(i, A_i, B_i)$ 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 $O(w2^w n)$ 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 nice DAG パス分解, 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 19 (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．Cal は各 i においてバッグ X_i を共通部分をもたない $k+1$ 個の頂点集合 $A_1,A_2,\ldots,A_k,B_i\subseteq X_i$ に分ける．その後 $X_1\cup X_2\cup\cdots\cup X_i$ によって誘導される部分グラフ G_i に対し，各 A_m $(m=1,2,\ldots,k)$ が s_m を始点とする点素パスとなっているかを考え，このような点素パスの組合せのうちパスの合計長が最小の組み合わせを求める．これを全ての $A_1,A_2,\ldots,A_k,B_i$ の組合せについて計算することで，G_i での合計長最小の点素パスの組合せを得る．全ての i について上記の計算を行うことで最終的に入力グラフ G の合計長最小の点素パスの組合せを得る.

Theorem 3. *DAG G に対し，幅が w である G の nice DAG パス分解が与えられたとき，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 20 (Cal). DAG $G=(V,E)$ の nice DAG パス分解を $X=(X_1,X_2,\ldots,X_s)$ 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 パス分解 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 7. Compute は G の *Disjoint Path Problem* の解を出力する．

Lemma 7の証明は付録で示す．

最後に Compute の計算量を示す．

Lemma 8. *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* パス分解 P が与えられたとき，$\mathsf{Compute}(P)$ 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，$|X_i| \le w+1$ 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 $0 \le i \le 2|V|+1$ である．さらに $\mathsf{pred}(v), \mathsf{suc}(v)$ の計算量がそれぞれ高々$O(w)$ であることに注意すると，$\mathsf{Cal}(i, \mathscr{A}_i, B_i)$ 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 nice DAG パス分解，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 21 (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 パス分解が与えられたとき，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．ST は各 i に

おいてバッグ X_i を共通部分をもたない頂点集合 $A_i, B_i \subseteq X_i$ に分ける．その後 $X_1 \cup X_2 \cup \cdots \cup X_i$ によって誘導される部分グラフ G_i に対して $R \cap V[G_i]$ をすべて含む有向シュタイナー木のうち，A_i を全て含み，B_i を全て含まず，かつサイズが最小の有向シュタイナー木を求める．これを全ての A_i, B_i の組合せについて計算することで，G_i での最小有向シュタイナー木を得る．全ての i について上記の計算を行うことで最終的に入力グラフ G の最小有向シュタイナー木を得る．以下では $d(e)$ を枝 e の重みとする．

Definition 22 (ST)**.** DAG $G = (V, E)$ の nice DAG パス分解を $P = (X_1, X_2, \ldots, X_s)$ 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 パス分解において 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 パス分解 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 パス分解に変換したものを，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 9. 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

Lemma 9の証明は付録で示す．

最後に Compute の計算量を示す．

Lemma 10. *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* パス分解 P，根 $r \in V$，$R \subseteq V$ が与えられたとき，$k = |R|$ とすると，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* パス分解が与えられたとき，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 DAG パス分解の利点

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 23 (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．Hanaka 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. 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 $4^{2tw^2}4^{2tw}n^{O(1)}$ 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幅が高々 tw の木分解を利用したアルゴリズムであり，DP を行う上での必要な情報として，木分解上の各バッグ B_t に対して部分解となる枝集合 $A\subseteq E(B_t)$，各頂点 $v\in B_t$ に対して前方の向きを支配できるかを表

す関数 f，f の支配関係の根拠を示す関数 s_f を保持する必要がある．こ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に対しても動作するといった利点がある．一方で木分解は入力グラフの枝の向きの情報をもたないため，向きの情報を表す f, s_f を計算する必要があり，計算量が $O(4^{2tw})$ だけ大きくなっている．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．このアルゴリズムで利用する DAG パス分解は入力グラフの向きの情報を内部に含んでいるため，f, s_f といったパラメータが不要になる．木幅が抑えられた DAG でも DAG パス幅は任意に大きくなることがあるため，計算量の単純な比較はできないが，木分解を用いた場合よりもアルゴリズムの構築が容易になる利点がある．★より詳しく読む

Theorem 6. *DAG G に対し，幅が w である G の nice DAG パス分解が与えられたとき，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 24 (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．以下の 2 つの補題により Theorem 6が示される．

Lemma 11. *DAG G に対し，幅が w である G の nice DAG パス分解が与えられたとき，幅が高々$O(w^2)$ である $L(G)$ の nice DAG パス分解を G の頂点数の多項式時間で構築できる．*

Lemma 12. *DAG G に対し，G の mDEDS と $L(G)$ の mDiDS は等しい．*

上記の 2 つの補題及び Theorem 6の証明は付録で示す．

第4章　DAGパス分解を求める$O(\log^2 n)$-近似アルゴリズム

一般のDAGに対するDAGパス幅を求める問題はNP困難であるが，近似的な幅であれば多項式時間で求めることができる．本章では頂点数n，最大出次数kであるDAG Gが与えられたとき，GのDAGパス幅に対して近似率が高々$O(k\log^2 n)$であるDAGパス分解を与える多項式時間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 25 (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$S_{\mathrm{alg},I}$，最適解を$S_{\mathrm{opt},I}$，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．

$$r = \sup_I \frac{f(S_{\mathrm{alg},I})}{f(S_{\mathrm{opt},I})}.$$

次にseparatorに関する定義と補題を与える．

Definition 26 (DAG edge-separator)**.** [19] 各枝が正の実数のコストを持つDAG Gに対し，SがGのDAG edge-separatorであるとは，$E[G]\backslash S$によって誘導される部分グラフが2つの独立な頂点集合A, Bによって構成されることであり，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 27 (b-balanced DAG edge-separator)**.** b-balanced DAG edge-separatorとは，各$A, B \subseteq V$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]で以下を示している．

Proposition 7. *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 $S \subseteq E[G]$が存在したとする．このときコスト$O(C(S)\log n)$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，DAG

パス幅の $O(\log^2 n)$ 倍以内の幅を持つ DAG パス分解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* パス幅が pw の *DAG* G が与えられたとき，幅が $O(k\ pw \log^2 n)$ の *DAG* パス分解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 Subroutine 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 パス分解である P_L, P_R を受け取り，G の nice DAG パス分解 P を返す Subroutine である (図**??**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．Theorem 7のアルゴリズムでは，Proposition 7のアルゴリズムで DAG を再帰的に分割し，複数の小さな頂点集合を得る．各頂点集合に対して nice DAG パス分解を得たのち，Merge によって DAG パス分解を結合していく．これを繰り返すことで最終的に元のグラフの nice DAG パス分解を得る．

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* パス分解 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 $(j = i, i+1, \ldots, s_l)$ 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．

Merge に関して以下の補題が得られる．証明は付録で示す．

Lemma 13. $P =$ Merge(P_L, P_R) は G の *nice DAG* パス分解である．

次に，DAG G が与えられたときに G の nice DAG パス分解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 パス分解 $P = (\varnothing, V, \varnothing)$ を返す．

2. Divide Step: A, B を，Proposition 7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．

ここで定義 7の証明のため，次の補題を示す．証明は付録で示す．

Lemma 14. 頂点数 n，最大出次数 k，*DAG* パス幅 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．

最後に次の補題を示すことで，定義 7が成り立つことを示す．

Lemma 15. 頂点数 n，最大出次数 k，*DAG* パス幅が 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 パス分解 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 パス分解 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，Proposition 7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 パス分解の幅が 0 であることに注意すると，Make が出力する nice DAG パス分解の幅 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* パス幅が pw の *DAG* G が与えられたとき，幅が $O(pw \cdot \log^{3/2} n)$ の *DAG* パス分解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．

Proposition 8. 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，定義 8を示すためには以下の補題を示せば十分である．

Lemma 16. *one-shot BP* と *nice DAG* パス分解は等価な問題である．

Lemma 16の証明は付録で示す．

第5章　$O(ld^t)$の幅のDAGパス分解を求めるアルゴリズム

本章では最大出次数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パス分解を出力するか，HのDAG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

Proposition 9. 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．

[8]による上記のアルゴリズムを参照し，最大出次数と根数がそれぞれd, lのDAG Hに対し，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同相埋め込みを利用する．

Definition 28 (同相埋め込み)**.** 有向グラフ$G_1 = (V_1, E_1)$の有向グラフ$G_2 = (V_2, E_2)$への同相埋め込みとは，以下の性質を満たすV_1からV_2への写像$f : V_1 \to V_2$である．

1. fは単射である．
2. E_1の各枝はG_2において互いに素なパスに対応する．

ここで互いに素なパスとは，有向グラフ上でそれぞれのパスが同じ枝や頂点を共有しないことを表す．このとき，G_2内のこれらの互いに素なパスは，G_1の枝の可能な分割を表す．ただし枝の分割とは，枝(u, v)を中間頂点wを含むパス$u \to w \to v$で置き換える操作を指す．

以下では同相埋め込みを単に埋め込みと表現する．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つの頂点集合X, Y $(X \cup Y = V[H], X \cap Y = \varnothing)$に分割できる．この

とき X, Y によって誘導される H の部分グラフをそれぞれ A, B とすると，H において A と B の間には A から B への枝しか存在せず，A の DAG パス幅は t より大きく $ld^{t+3} - 1$ より小さい．

まず，Theorem 9を示す際に必要となる完全有向木を定義する．

Definition 29. ある正整数 d に対し，有向木 $T = (V, E)$ が以下をすべて満たすとき，T は完全有向 d 分木である．

1. 葉を除く任意の頂点の出次数が d である．
2. 根から全ての葉までのパスの長さが等しい．

T が完全有向 d 分木であるとき，根から葉までのパスの長さを h とし，$h + 1$ を T の高さとする．T が根のみからなるグラフの場合，T の高さは 1 となる．

Lemma 17. $T_{h,d}$ 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 DAG パス幅は明らかに 1 であるため補題を満たす．次にある $h > 1$ での補題の成立を仮定すると，$T_{h,d}$ に幅が $h - 1$ の DAG パス分解 X_h が存在する．ここで $T_{h+1,d}$ は d 個の $T_{h,d}$ の根と単一の根 r を夫々接続したグラフであることに注意すると，幅が h である $T_{h+1,d}$ の DAG パス分解を以下のように構成できる．はじめに r のみからなるバッグを用意し，次に d 個の DAG パス分解 X_h に対し，X_h の各バッグに r を追加したものを順に d 個つなげる．こうしてできる頂点集合の列は DAG パス分解の 3 つのルールを満たすため $T_{h+1,d}$ の DAG パス分解であり，幅は h である．また幅が h より小さい DAG パス分解は存在しない．なぜならば，$T_{h+1,d}$ は $T_{h,d}$ を内部に含むため，$T_{h+1,d}$ の DAG パス幅は $h - 1$ より小さくならない．ここで幅が $h - 1$ である $T_{h+1,d}$ の DAG パス分解の存在を仮定する．このとき d 個の $T_{h,d}$ は互いに到達不可能であるため，それぞれ独立して DAG パス分解を構成でき，このとき d 個の $T_{h,d}$ の DAG パス分解を並列に行うよりも幅が大きくならないことに注意する．d 個の $T_{h,d}$ のうち最初に DAG パス分解を構成するものを $T'_{h,d}$ とすると，DAG パス分解のルール 2 より $T_{h+1,d}$ の任意の DAG パス分解は必ず最初のバッグに根 r を含むため，$T'_{h,d}$ の DAG パス分解で幅が $h - 1$ で最大となるバッグ (X' とする) には r を含んでいない．DAG パス分解のルール 3 より r は X' 以降で再びバッグに現れないため，$T'_{h,d}$ 以外で r と接続する $d - 1$ 個

の $T_{h+1,d}$ は存在しないことが必要．すなわち $d=1$ であることが必要だが，これは $d>1$ であることと矛盾する．X' より前に r に接続する頂点を全てバッグに含めていた場合でも，X' で必ず $T'_{h,d}$ 以外の頂点が含まれるため，幅が $h-1$ より大きくなり矛盾する．以上，背理法により幅が $h-1$ である $T_{h+1,d}$ の DAG パス分解は存在せず，$T_{h+1,d}$ の最適な DAG パス分解の幅は h である．よって h でも補題が成り立つため，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，根が 1 つとなるように H に頂点と枝を加える．具体的には高さが $\lceil \log_d l \rceil$ である完全有向 d 分技を用意し，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．H' は出次数が高々dであり，根の数が 1 である．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' への埋め込みを見つける．もし埋め込み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．アルゴリズムでは $M_{t,d,l}$ の頂点を H' 上に貪欲に埋め込んでいく．ここで $M_{t,d,l}$ の頂点をトークンと呼ぶ．可能な限り埋め込みを行ったあと，H' 上に置かれたトークンを他の頂点上に移すことでさらに埋め込みを進める．$M_{t,d,l}$ の全トークンが埋め込みに使われた場合は $M_{t,d,l}$ から H' への埋め込みを発見したことを表す．トークン 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．

アルゴリズムでは H' の頂点に $M_{t,d,l}$ のトークンを配置していく操作を繰り返す．このとき $M_{t,d,l}$ から H' への埋め込みの条件を満たすようにトークンを配置することに注意する．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定し，H' のある頂点 v にトークンが置かれた場合，v の色

Algorithm 1 GrowTokenTree

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**
    place token $T \cdot b$ on v
end while
return {tokened vertices of H'}

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．

GrowTokenTree と FindEmbedding の擬似コードをそれぞれ Algorithm 1, 2で示す．GrowTokenTree (Algorithm 1) では，$M_{t,d,l}$ の木構造を保ったまま貪欲に H' の頂点にトークンを配置していく．このときトークンを置くことができる H' の頂点は，先行頂点に全てトークンが置かれている頂点のみである．この条件を満たす頂点がなくなるまで H' にトークンを配置していき，最後にトークンが置かれた H' の頂点集合を出力する．

FindEmbedding (Algorithm 2) ではDAGパス分解となる頂点集合の列 $(X_1, X_2, \dots)$ の出力を試みる．最初に H' のただ一つの根にトークン λ を置く．その後 $i = 1$ とし，GrowTokenTree を実行して出力結果を X_1 とする．次に各 i について以下を繰り返す．$(X_1, X_2, \dots, X_i)$ までの構築ができているとする．X_i で $M_{t,d,l}$ の全てのトークンを同時に使用している場合，これは $M_{t,d,l}$ から H' への埋め込みを表しているため，H' の DAG パス幅は $M_{t,d,l}$ の DAG パス幅より小さくならないことがわかる．また H' の全ての頂点が red になった場合，H' の各頂点に 1 度だけトークンを配置したことになる．このとき列 $(X_1, X_2, \dots, X_i)$ は H' の DAG パス分解になっており，証明はこの後に示す．上記の 2 つの場合以外，すなわち $M_{t,d,l}$ の全てのトークンを同時に使用しておらず，かつ H' に少なくとも 1 つの blue の頂点が存在する場合，トークンの配置を変更することでさらに GrowTokenTree の処理を進めることができる可能性がある．したがって以下で述べる条件を満たす頂点 $v \in V[H']$ について，v に置かれたトークン T を v か

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

ら取り除くことを考える．T を v から取り除くことができる条件は，(a)v の後続頂点が全て red であり，かつ (b)$M_{t,d,l}$ 上で T の配置済みの子トークンが高々 1 つしかないことである．前者は DAG パス分解において forget が可能な条件に対応し，後者は埋め込みの状態を保つための条件に対応する．トークン T が上記の 2 つの条件を満たしている場合のみ，T を v から取り除くことができる．このとき $M_{t,d,l}$ 上で配置済みのトークン集合を取り出すと T で非連結となる場合があるため，配置済みの T の子，孫をそれぞれ T, T の子と置き換えることで，$M_{t,d,l}$ 上で配置済みのトークン集合が常に連結になるようにできる．この操作に

よって T の孫が未配置となった場合，孫を H' の別の頂点に配置できる可能性があるため，再び GrowTokenTree を行い，X_{i+1} をその出力とする．条件 (a)(b) を共に満たすトークンが存在しない場合，これ以上トークンの配置を進めることはできず，最後に行った GrowTokenTree の出力をそのまま返してアルゴリズムは終了する．このとき H' の DAG パス幅は $M_{t,d,l}$ の DAG パス幅より小さくならず，これも以下で証明する．

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 パス分解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 パス分解のルール 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トークン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 パス分解のルール 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 からトークンは削除されず，また GrowTokenTree の while の条件より，v の先行頂点にはすべてトークン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 パス分解のルール 3 を満たす．以上より頂点集合の列 $X_{H'}$ は H' の DAG パス分解となっている．$\lceil \log_d l \rceil < \log_d l + 1$ に注意すると，このとき $X_{H'}$ 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 パス分解の 3 つのルールを満たすことから H の DAG パス分解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 パス分解が得られる．

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 パス分解になっている．まず A' の定義より明らかに DAG パス分解のルール 1 を満たす．また上記と同様の議論により DAG パス分解のルール 2, 3 を満たす．よって $X_{A'}$ は A' の DAG パス分解である．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 パス分解の 3 つのルールを満たすことから A の DAG パス分解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 パス分解が得られる．

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' への埋め込み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，明らかに埋め込みの条件を満たす．したがって FindEmbedding の列 6 から列 12 において埋め込み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 $M_{t,d,l}$ の各頂点に対して埋め込み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．したがって埋め込み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埋め込み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．したがって埋め込みの

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' への埋め込みを表している．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 への埋め込みが存在する．Lemma 17より T_A の DAG パス幅は $t+1$ であるため，A の DAG 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 DAG 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 DAG 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 パス分解において forget することはできない．なぜならば DAG パス分解のルール 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 パス分解において 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 DAG 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 DAG 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 への埋め込みを出力するか，幅が高々$O(ld^t)$ の DAG 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木幅

6.1　DAG木幅とDAG木分解の定義

本節ではDAGパス幅の拡張としてDAG木幅・DAG木分解を導入する．

Definition 30 (DAG木分解)**.** 有向グラフ $G = (V, E)$ のDAG木分解とは，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 r に対し，任意の枝 $(u, v) \in E$ について $u, v \in \beta(r)$ か，もしくはある i $(i \in T, i \neq r)$ とその親 j があり，$u, v \in \beta(i)$, $v \notin \beta(j)$ が成り立つ．
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木分解の例を図??で示す．以下でDAG木幅を定義する．

Definition 31 (DAG木幅)**.** あるDAG木分解 (T, β) の幅を $\max_{t \in T} |\beta(t)|$ と定義する．このとき，有向グラフ G のDAG木幅とは，G に対する全てのDAG木分解のうち，幅が最小であるDAG木分解の幅である．

以降では，G のDAGパス幅, DAG木幅をそれぞれ $\mathsf{pw}(G), \mathsf{tw}(G)$ と表す．DAG木幅を求める問題の計算量クラスについて，以下を示す．

Theorem 10 (計算量クラス)**.** 有向グラフ G が与えられたとき，G の *DAG* 木幅を計算する問題は *NP* 困難である．

Theorem 10の証明は付録で示す．

最後にnice DAGパス分解と同様にnice DAG木分解についても定義する．

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

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 $S \subseteq V$ があり，$S \cap \beta(i) = \varnothing$ かつ $\beta(j) = \beta(i) \cup S$
   - **forget**　i がただ1つの子 j をもち，ある $v \in V$ があり，$\beta(j) = \beta(i) \backslash \{v\}$
   - **union**　i がちょうど2つの子 j, k をもち，$\beta(i) = \beta(j) = \beta(k)$

nice DAG 木分解の例を図??で示す．ある DAG 木分解が与えられたとき，幅が同じである G の nice DAG 木分解を効率よく構築することができる．

Theorem 11 (nice DAG 木分解の効率的な構築)**.** (T, β) が有向グラフ $G = (V, E)$ の *DAG* 木分解であるとする．また (T, β) の幅を w とする．このとき，幅が w であるような G の *nice DAG* 木分解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 パス分解を得る．

1. T の根に親 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，$\beta(i) \not\subseteq \beta(j)$ かつ $\beta(j) \not\subseteq \beta(i)$ 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 をもつとき，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) とつなぐ.

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

6.2 co-DAG 木幅

DAG 木分解 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 co-DAG 木幅を定義する．そのためにまず co-DAG 木分解の定義を行う．

Definition 33 (co-DAG 木分解)**.** 有向グラフ $G = (V, E)$ の co-DAG 木分解とは，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．

DAG 木分解は枝の向きと同方向に向かってバッグを構成していたが，co-DAG 木分解は枝の向きと逆方向に向かってバッグを構成している．これによって co-DAG 木分解を FPT アルゴリズムに利用する際，葉側のバッグから動的計画法を実行することができる．以下では co-DAG 木幅を定義する．

Definition 34 (co-DAG 木幅)**.** ある co-DAG 木分解 (T, β) の幅を $\max_{t \in T} |\beta(t)|$ と定義する．このとき，有向グラフ G の co-DAG 木幅とは，G に対する全ての co-DAG 木分解のうち，幅が最小である co-DAG 木分解の幅である．

以降では，G の co-DAG 木幅を $\mathsf{ctw}(G)$ と表す．co-DAG 木幅を求める問題の計算量クラスについて，以下を示す．

Theorem 12 (計算量クラス)**.** 有向グラフ G が与えられたとき，G の *co-DAG* 木幅を計算する問題は *NP* 困難である．

Theorem 12の証明は付録で示す．最後に nice co-DAG 木分解についても定義する．

Definition 35 (nice co-DAG 木分解)**.** $G = (V, E)$ を有向グラフとし，(T, β) を G の co-DAG 木分解とする．(T, β) が以下の 2 つを満たすとき，(T, β) は G の nice co-DAG 木分解であるという．

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 $S \subseteq V$ があり，

$S \cap \beta(j) = \varnothing$ 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 木分解が与えられたとき，幅が同じである G の nice co-DAG 木分解を効率よく構築することができる．

Theorem 13 (nice co-DAG 木分解の効率的な構築)**.** (T, β) が有向グラフ $G = (V, E)$ の *co-DAG* 木分解であるとする．また (T, β) の幅を w とする．このとき，幅が w であるような G の *nice co-DAG* 木分解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 パス分解を得る．

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, 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 木分解は，幅が高々w であり，また G の nice co-DAG 木分解である．この構築は G の頂点数の多項式時間で行える． □

第7章　co-DAG木幅を用いた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木分解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木分解が与えられたとき，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 36 (DS)**.** DAG $G = (V, E)$の nice co-DAG木分解を(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 木分解 (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 18. Compute は G の *mDiDS* のサイズを返す．

18の証明は付録で示す．

最後に Compute の計算量を示す．

Lemma 19. *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* 木分解 (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DAG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).