時間変化を考慮したネットワーク信頼性評価

研究代表者 中畑裕 奈良先端科学技術大学院大学先端科学技術研究科助教

1 背景

光通信網や携帯電話通信網などのネットワークは我々の生活に必要不可欠なインフラストラクチャであり，故障することなく常に動作し続けることが求められる.しかし,ネットワークの構成要素であるノードとノードをつなぐリンクは,災害等のため故障したり,断線したりすることがある.こうした故障の発生確率をゼロにすることはできないため，ネットワークは構成要素が故障することを前提として設計される必要がある.その際に重要となるネットワークの故障に対する強さを定量化した尺度の一つとして，ネットワーク信頼性[1]が挙げられる.ネットワーク信頼性は，ノードは完全に機能するが，リンクが確率に従って独立に故障するとき,指定された2ノード(始点と終点)間で通信が行える確率と定義される.ネットワーク信頼性は通信だけでなく，コンピュータアーキテクチャの故障に対する強さや物流事業，送電事業などのサービスの質に対する重要な指標であり,60年以上にわたって研究されてきた[1,2].ネットワーク信頼性を計算する問題は#P困難で[3, 4]，理論上効率よく計算することは難しいが，これを効率よく計算するために，これまで様々な試みがなされてきた.その一つとして，二分決定グラフ(binary decision diagram; BDD)[5]を用いた効率のよい信頼性計算法[6, 7]が知られている.しかし，それらネットワーク信頼性の計算手法の多くは，すべてのリンクが同時に故障する静的なモデルを考慮しており,ネットワークの時間変化は考慮していない.一方,近年，通信機器の高性能化と小型化，低廉化に伴って，宇宙ネットワークや車両アドホックネットワーク，ドローンネットワークなどの移動体で構成され，ネットワークトポロジーが時間変化するようなネットワークが身近なものになりつつある[8].ネットワークトポロジーが時間変化するネットワークをTVN(time-varying network)と呼び，この信頼性を考える事例として，スペースミッションなどのミッションクリティカル系のものが挙げられるため，TVN の信頼性(TVN 信頼性)の効率のよい計算手法が望まれている.TVNは各辺が時刻ラベル(正の整数)をもつ時間的グラフ(temporal graph)[9]でモデル化される.このとき，TVN 信頼性は，ノードは完全に機能するが，リンクが確率に従って独立に故障するとき，始点を出発したデータパケットが，時刻ラベルが単調増加する経路，即ち，旅路(journey)[9]を辿り，終点にたどり着く確率と定義される.Chaturvedi らは TVN信頼性を計算するために，まず経路を陽に列挙し，その中から旅路を抽出,そして sum of disjoint products (SDP)法[10]を用いて,信頼性を計算するアルゴリズムを提案した[11].しかし，この手法には計算量が頂点数に対して，指数的に増加するという欠点がある. 一方，従来から考えられてきた静的なモデルのネットワーク(静的ネットワーク）の信頼性に対しては，BDDを用いた効率のよい計算法があることを先に述べた.そこで本研究では，この効率の良い手法を TVN向けに拡張することで,TVN信頼性計算の高効率化を図ることを目的とする.

2 準備

本章では，時間的グラフと旅路について説明した後，TVN 信頼性を定義する. そして，提案手法で用いられる二分決定グラフ（binary decision diagram; BDD）と呼ばれるデータ構造と，グラフを列挙するアルゴリズムの一種であるフロンティア法について説明する.

2-1 時間的グラフと旅路

時間的グラフは各辺に時刻ラベルが付与された無向グラフである［9］. $G = (V, E, t)$ を時間的グラフとする. ただし，Vを頂点集合とし，$E = \{e_1, \ldots, e_m\}$ を無向辺の集合とする（多重辺を許す）. t: $E \to \mathbb{N}$（$\mathbb{N}$は正の整数の集合）を時刻ラベルとする. 各辺はただ 1つの時刻ラベルが割り当てられ，その時刻ラベルの時刻にのみ存在する. 時間的グラフでは，頂点集合は時間変化しないが，辺集合が時間変化し，連結成分が時間変化する.

次の条件すべてを満たす辺列 $S = (e_{i1}, \ldots, e_{ik})$ を考える.

- $\forall j \in \{1, \ldots, k-1\}$,（$e_{ij}$と e_{ij+1} は端点を共有）$\land$（$t(e_{ij}) \le t(e_{ij+1})$）

公益財団法人電気通信普及財団
研究調査助成報告書 第38号 2023年度

- $e_{ij} \neq e_{ij'}$（ただし，j, j' を $j < j'$ を満たす 1 以上 k 以下の正の整数とする）

S において，u を e_{i2} と端点を共有しない e_{i1} の端点とし，v を e_{ik-1} と端点を共有しない e_{ik} の端点とする．このとき，S を u–v 旅路（あるいは単に旅路）と呼び，ここでは辺列 S の各項を要素にもつ集合も旅路と同一視し J と表記する．S の連続な部分列もまた旅路であり，これを部分旅路（subjourney）と呼ぶ．また，時刻ラベルが広義単調増加する旅路をマルチホップ（multi-hop）型旅路，狭義単調増加する旅路をシングルホップ（single-hop）型旅路と呼ぶ．

時間的グラフ G において，u から v までの旅路が存在するとき，ここでは，u は v に連絡するといい，$u \rightsquigarrow_G v$ と表記する．u から v までの旅路が存在しないときは，u は v に連絡しないといい，$u \not\rightsquigarrow_G v$ と表記する．$u \rightsquigarrow_G v \Longrightarrow v \rightsquigarrow_G u$ は必ずしも成立しない．

図 1 に時間的グラフの例を示す．ノード間のつながりが時間変化するネットワーク，即ち，TVN のモデル化のためには，連結成分の時間変化の特徴を捉える必要があるため，この時間的グラフが用いられる．

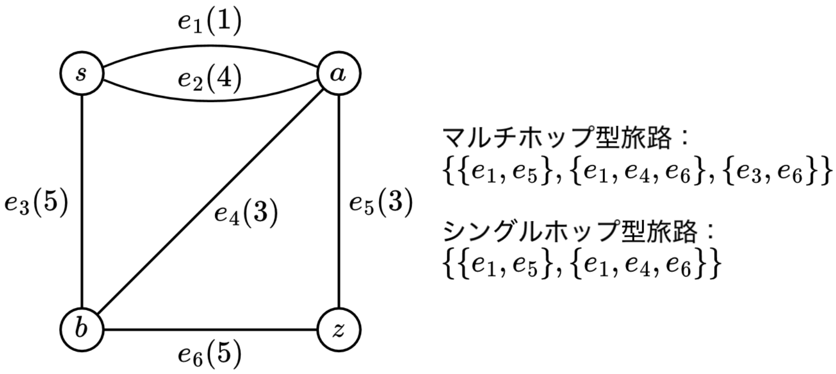


図 1: 時間的グラフの例．括弧内の数字は時刻ラベルを示す．例として，マルチホップ型，シングルホップ型の s–z 旅路を図右にそれぞれ示す．

2-2 TVN 信頼性

TVN は時間的グラフ $G = (V, E, t)$ を用いて表現される（以後，TVN と時間的グラフ G を同一視する）．ネットワークを構成するノードはグラフの頂点に対応し，リンクは辺に対応する．メッセージなどのデータの送信者を表す頂点を始点と呼び，送信先を表す頂点を終点と呼ぶ．始点を s とし，終点を z とする．$X \subseteq E$ の各要素（辺）の端点を集めた集合を $V[X]$ とする（$V[X] := \cup_{\{u,v\} \in X}\{u, v\}$）．また，$X$ による誘導部分グラフを $G[X]$ とする．$G[X] := (V[X], X, t')$ もまた時間的グラフである．ただし，$t' : X \to \mathbb{N}$ は任意の $e \in X$ に対し，$t'(e) = t(e)$ であるような時刻ラベルである．$X \subseteq E$ が $s \rightsquigarrow_{G[X]} z$ を満たすとき，X を始終点連絡辺集合と呼ぶ．

TVN 信頼性は，各辺がある確率で独立に削除された後でも，終点が始点から到達可能となっている確率である．時間的グラフ G の信頼性を $\sigma(G)$ とし，$\mathcal{S}_G \subseteq 2^E$ を，G に対する始終点連絡辺集合すべてからなる集合族とする．$e_i \in E$ が削除される確率を $q(e_i)$ とし，$p(e_i) = 1 - q(e_i)$ を生存確率とすると，TVN 信頼性は，

$$\sigma(G) := \sum_{X \in \mathcal{S}_G} p(X) \quad (1)$$

と定義される．ただし，

$$p(X) := \prod_{e_i \in X} p(e_i) \prod_{e_j \in E \setminus X} (1 - p(e_j)) \quad (2)$$

とする．

TVN 信頼性計算は，入力として時間的グラフ G が与えられたとき，その時間的グラフの信頼性 $\sigma(G)$ を計算する問題である．マルチホップ型旅路を許す場合の TVN 信頼性（マルチホップ TVN 信頼性）は，すべての辺の時刻ラベルを 1 とすると，静的ネットワーク信頼性と同義である．静的ネットワーク信頼性計算は#P

困難である［3,4］から，マルチホップ TVN 信頼性計算問題は#P 困難である．シングルホップ型旅路のみ許す場合の TVN 信頼性（シングルホップ TVN 信頼性）計算問題は，マルチホップ TVN 信頼性計算問題と異なり，静的ネットワーク信頼性計算問題の一般化になっていないため，直ちに#P 困難であるとはいえない．ただし，現状，シングルホップ TVN 信頼性計算問題に対する多項式時間アルゴリズムは知られていない．また，関連する問題である，各辺が有向枝の時間的グラフの強連結成分を求める問題は NP 困難で［12］，その数え上げは#P 困難だと考えられることから，シングルホップ TVN 信頼性計算問題も同程度の難しさだと考えられる．そこで，始終点連絡辺集合を陽に列挙することを避け，これを陰に列挙するアプローチをとることで，TVN 信頼性を効率よく計算する方法を考える．

2-3 二分決定グラフ

始終点連絡辺集合を陰に列挙するために，二分決定グラフ（binary decision diagram; BDD）［5］を用いる．BDD は，Shannon の展開定理に基づき，ブール関数をコンパクトに表現したものである．ブール関数は集合族を表現するための指示関数としても用いられる．ここでは，辺集合 E の要素を変数とし，部分集合 $X \subseteq E$ に対して，$e_i \notin X$（$e_i \in X$）ならば $False$（$True$）を返すような指示関数として BDD を用いる．

BDD は，節点集合 N と有向枝の集合 A からなる，根付き有向非巡回グラフ $B = (N, A)$ である．BDD はどの有向枝にも指されない節点（根節点）ρ がちょうど 1 つと，他の節点を指す有向枝をもたない節点（終端節点）として，ちょうど 2 つの節点，$\bot$-終端節点と $\top$-終端節点をもつ．終端節点以外の節点 $\alpha \in N$ は変数ラベル $l(\alpha) \in \{1, ..., m\}$ をもち，他の節点を指す有向枝として，ちょうど 2 つの有向枝，0-枝と 1-枝をもつ．α の x-枝（$x \in \{0, 1\}$）に指される節点を x-子節点と呼び，α_x と表記する．α を α_x の親節点と呼ぶ．ただし，α_x が終端節点でないとき，$l(\alpha) < l(\alpha_x)$ とする．

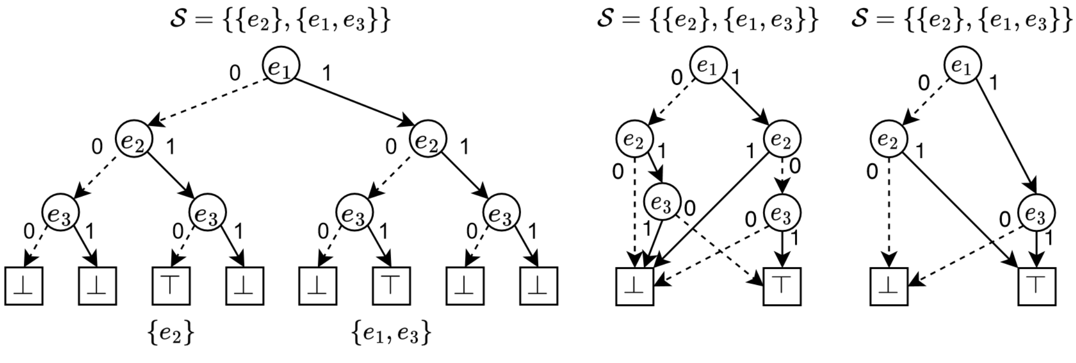
ρ から $\top$ までの有向経路はブール関数が $True$ であるような変数割当てを表している．有向経路が α の 0-枝（1-枝）を辿るならば，変数 $e_{l(\alpha)}$ には $False$（$True$）が割当てられる．またこのとき，$e_{l(\alpha)}$ に $True$ を割当てることを採択するという．

BDD は以下の 2 つの簡約規則を二分決定木（binary decision tree）に対して可能な限り適用した結果として得られる．

1. 節点削除規則：$\alpha_0 = \alpha_1$ ならば，α を削除し，α を指す枝すべてが $\alpha_0 (= \alpha_1)$ を指すようにする
2. 節点共有規則：$l(\beta) = l(\beta')$ の節点で，$\beta_0 = \beta'_0$ かつ $\beta_1 = \beta'_1$ ならば，β と β' を共有

以上 2 つの規則を可能な限り適用すると冗長な節点がなくなり，最終的に得られた BDD は既約であるという．既約な BDD は一意に定まる［5］．また $\alpha \neq \rho$ を根節点とする部分グラフもまた BDD である．以後，便利のため，BDD の節点数を$|B|$と表記する．

BDD は密な集合族を少ない節点数で表現することができるという利点がある［13］．一方，疎な集合族の表現に適したデータ構造として，ゼロサプレス型 BDD（zero-suppressed BDD; ZDD）［14］が挙げられる．ZDD は BDD から派生したデータ構造で，BDD と同様，根付き有向非巡回グラフ $Z = (N, A)$ である．



(a) 二分決定木　(b) BDD　(c) ZDD

図 2: 二分決定木と BDD と ZDD．実線の矢印を 1-枝，破線の矢印を 0-枝とする．

ZDD も二分決定木に対し簡約規則を可能な限り適用した結果得られるが，以下のように節点削除規則が BDD の場合と異なる．

1. 節点削除規則：$\alpha_1 = \bot$ ならば，α を削除し，同時に α を指していた有向枝に α_0 を指させる
2. 節点共有規則：$I(\beta) = I(\beta)$の節点で，$\beta_0 = \beta'_0$かつ$\beta_1 = \beta'_1$ならば，β と β を共有

以後，便利のため，ZDD の節点数を |Z| と表記する．

図 2 に集合族 $S = \{\{e_2\}, \{e_1, e_3\}\}$を表す二分決定木と BDD，そして ZDD を示す．集合族が疎なため，BDD と比べ ZDD は節点数が少ない．

2-4 フロンティア法

SG を表す BDD 構築のために，フロンティア法（frontier-based search; FBS）[15] と呼ばれる手法を用いる．フロンティア法は所望の条件を満たす集合を陰に列挙するための汎用的なフレームワークである．本節ではフロンティア法のフレームワークについて説明する．

関数 $C\colon 2^E \to \{0, 1\}$ を考える．この関数は $X \subseteq E$ を入力とし，0 か 1 を出力する．$C(X) = 1$ ならば，X は性質 C をもつという．$P = \langle G, C \rangle$ を性質 C をもつ E の部分集合すべてを得る問題とする．このとき，P の解 EP は

$$\mathrm{E}P := \{X \in 2^E \mid C(X) = 1\} \quad (3)$$

と定義される．問題 Pが与えられたとき，フロンティア法のアルゴリズムは辺を順番に処理していく．辺を処理するとは，節点の集合 $N_i := \{\alpha \mid I(\alpha) = i\}$ を $i = 1, \ldots, m$ に対して構築することと，$x \in \{0, 1\}$ に対して，x-枝の集合 $A_x := \{(\alpha, \alpha_x) \mid \alpha \in N ¥ \{\bot, \top\}\}$ を構築することをいう．i 番目の辺を処理しようとするときの処理済みの辺集合を $E_{\leq}^{i-1} := \{1\ e_1, \ldots, e_i{-}1\}$ とし，未処理の辺集合を $E_{\geq}^{i} := \{e_i, \ldots, e_m\}$ とする．E$(\alpha) \subseteq 2^{E_{\leq}^{i-}}$ を根節点から $\alpha \in N_i$ までの経路に対応する辺の部分集合とする．各節点 $\alpha \in N_i$ は P の部分問題 $P_\alpha := \langle G[E_{\geq}^{i}], C_\alpha \rangle$ と対応している．ただし，C_α を，

$$C_\alpha(X) = 1 \iff \forall Y \in \mathrm{E}(\alpha), C(X \cup Y) = 1 \quad (4)$$

と定義される性質とする．任意の節点のペア $\beta, \beta \in N_i$ は，任意の $X \in 2^{E_{\geq}^{i}}$ に対して，$C_\beta(X) = 1 \iff C_\beta(X) = 1$ が成立するならば，等価であるという．フロンティア法のアルゴリズムは，複数の等価な節点を 1 つにまとめる．

フロンティア法のアルゴリズムは，最初に根節点のみからなる節点集合 $N_1 = \{\rho\}$ をつくる．i 段階目では，N_i から次の手順で N_{i+1} をつくる．各 $\alpha \in N_i$ に対して，部分集合族 E(α) から e_i を含まない部分集合からなる族 E(α_0) と e_i を含む部分集合からなる族 E(α_1) の子節点をつくる．新たな子節点の生成時，節点数の増加を抑えるため，以下の手続きを実行する．

- 枝刈り：関数 Prune(α, e_i, x)が $True$ を返すならば，α の x-子節点を⊥とし，x-枝（$\alpha, \bot$）を A_x に加える．
- マージ：β を α の子節点とする．β と $\beta \in N_{i+1}$ が等価ならば，β を β とする．

これらの手続きを効率的に行うために，βは補助的な情報としてコンフィギュレーション $\phi(\beta)$を保持する．コンフィギュレーションは$\phi(\beta) = \phi(\beta)$ならば，$\beta$ と β は等価であるという条件を満たす．この逆は必ずしも成立せず，冗長な節点の生成の原因となることに留意する．ただし，冗長な節点とは，任意の $X \in 2^{E_{\geq}^{i}}$ に対して $C_\beta(X) = 1 \iff C_\beta(X) = 1$ を満たすが，$\phi(\beta) \neq \phi(\beta)$ であるような節点のことをいう．

枝刈りとマージを行いながら，辺を処理していくと，やがて最後の辺 e_m を処理することとなる．このときの節点を α_m とする．Prune$(\alpha_m, e_m, x) = False$ ならば，α_m の x-子節点を $\top$ とし，x-枝（$\alpha_m, \top$）を A_x に加える．ここでは，x-枝($\alpha_m, \top$) を A_x に加えることを受理すると呼ぶ．

本質的に，フロンティア法はコンフィギュレーションを状態としてもつ動的計画法である．フロンティア法のアルゴリズムは，動的計画法の計算結果をメモした表をもとに，BDD を構築する．

3 提案手法

本章では TVN 信頼性を計算するアルゴリズムを提案する．

3-1 提案手法の流れ

提案手法の大まかな流れを以下に示す．

Step 1 旅路の集合を表す ZDD（旅路 ZDD）を構築

Step 2 旅路 ZDD をもとに始終点連絡辺集合の集合族 SG を表す BDD を構築
Step 3 得られた BDD に対して動的計画法を適用し，TVN 信頼性 $\sigma(G)$ を計算

提案手法は，入力から SG を表す BDD B を直接構築するのではなく，旅路 ZDD を経由して B を構築するアプローチをとる．この理由は，もし入力から B を直接構築する場合，フロンティア法はコンフィギュレーションのチェックと更新を行いながら辺を処理していくが，これにかかる計算量が旅路 ZDD を構築する場合と比べ大きく，また計算途中の節点数が旅路 ZDD を構築するときより，爆発的に大きくなると予測されるためである（詳細は付録 A に述べる）．そこで，旅路 ZDD を経て B を得るアプローチをとることにした．また旅路列挙に際して，ZDD を利用するのは，旅路の集合が疎であることが予測されるためである．一方，SGの表現に BDD を利用するのは SG が密であることが予測され，BDD を利用することで節点数が抑えられると期待されるためである．Step 1 の旅路 ZDD を構築する方法は現時点では知られておらず，本研究の主な技術的貢献となる．Step 2，3 は既存手法を用いて，以下のように可能である．

始終点連絡辺集合は，辺の部分集合で，旅路が存在するもの，即ち，始点から終点までの旅路を少なくとも 1 つ含むような辺の集合のことであった．よって，その集合族は，

$\mathrm{S}G := \{U \subseteq E \mid \exists J \in \mathrm{J}, J \subseteq U\}$ (6)

と定義される．ただし，旅路の集合を J とする．ここでは，集合 Jが集合 Uの部分集合であるとき，Uは Jの上位集合であるという．提案手法では，旅路 ZDD を構築後，superset 演算 [17] をその ZDD に適用することで，SG を表す BDD を構築する．Superset 演算の計算量は，セパレータ（separator）というものを導入することで評価される．ZDD Z の i 番目のセパレータとは，$j \leq i < k$ を満たす正の整数 j をラベルにもつ節点を始点，k をラベルにもつ節点を終点とする有向枝をもつような，終端節点以外の Z の節点の集合のことである．セパレータサイズの最大値を sep(Z)とすると，superset 演算の計算量は $O(|\mathrm{Z}| \cdot |E|^2 \cdot 2^{2\mathrm{sep}(\mathrm{Z})})$ である．

SG の BDD B = (N, A) が得られたら，ボトムアップ的に動的計画法を B に適用することで TVN 信頼性を計算できる [6,7]．各節点 $\alpha \in N$ に，α の子孫が表す部分グラフの確率の総和 $\psi(\alpha)$ を保持させる．⊥ と ⊤ の保持する確率の値は，そ
れぞれ $\psi(\bot) = 0$，$\psi(\top) = 1$ とする．終端節点でない節点 $\alpha \in N$ の $\psi(\alpha)$ は，

$\psi(\alpha) = \psi(\alpha_1)p(e_{l(\alpha)}) + \psi(\alpha_0)(1 - p(e_{l(\alpha)}))$ (7)

と計算される．(7)より，$\sigma(G)$と $\psi(\rho)$は等しく，その計算にかかる時間は $O(|\mathrm{B}|)$ である．

3-2 旅路列挙のためのフロンティア法

旅路を列挙するために，2.4 節で述べたフロンティア法の 3 つの主要構成部分，即ち，コンフィギュレーション，枝刈り，GenerateNode 関数を設計することで，旅路列挙のためのフロンティア法（FBS for journey enumeration；FBSJE）のアルゴリズムを構築する．

始点 s から終点 z までの旅路を得ることが目的であるため，

$C(X) = 1 \iff X$ は G の s–z 旅路である (8)

と定義する．

コンフィギュレーションここでは旅路の集合 J を列挙するためのコンフィギュレーションを設計する．最初に旅路の列挙に必要となるコンフィギュレーションを説明し，次にそのコンフィギュレーションがフロンティア法を利用するために必要な条件を満たすことを示す． $i = 1,...,m$ に対して，

$F_i := V[E^{\leq i-1}] \cap V[E^{\geq i}]$ (9)

を i 番目のフロンティアと定義する．フロンティアは処理済みの辺と未処理の辺の両方が接続する頂点集合である．

既存の経路列挙のためのフロンティア法 [15] は連結成分と次数の 2 つの情報をコンフィギュレーションとしてフロンティアの頂点に保持させる．これを旅路列挙に対応させるために，連結成分と次数に加え，時刻ラベルの合計 3 つの情報をコンフィギュレーションとしてフロンティアの頂点に与える．具体的には $\alpha \in N$に対し，各行が連結成分（comp），次数（deg），時刻ラベル（time）を表し，列にフロンティアの頂点 vを添字にもつ行列 $\phi(\alpha)$ をコンフィギュレーションとする．ただし，可読性のため，コンフィギュレーションを，フロンティアの頂点 vを添字とする連結成分の配列 comp_α，次数の配列 deg_α，時刻ラベルの配列 time_αにわけて表現する．α の添字のついたコンフィギュレーションは $\alpha \in N_i$ のコンフィギュレーションであることを意味する．節点 α について，comp (v) はフロンティアの頂点 $v\,F$ の連結成分を記録し， $\mathrm{deg}_\alpha(v)$は次数を記録する．$\mathrm{time}_\alpha(v)$については，v が旅路の端点であるときに限り，その時刻ラベルを記録することを

約束する．

ここで，FBSJE のアルゴリズムは，フロンティアに含まれる始点と終点が孤立点か旅路の端点のいずれかであること，それ以外の頂点については孤立点か，旅路の内点か，旅路の端点かのいずれかであることを保証するように枝刈りを実行するものと仮定する．このとき，コンフィギュレーションを利用して，(4) を満たす性質を定義できれば，コンフィギュレーションがフロンティア法を利用するのに必要な条件，即ち，コンフィギュレーションが同じならば，処理済みの辺の採択の仕方に関わらず，未処理の辺をどう採択したら解となるかが同じであることが示される．以下に (4) を満たす性質を定義する．

$\alpha \in N_i$ に対して，次のように時間的グラフ $G_\alpha = (V_\alpha, E_\alpha, t_\alpha)$ をつくる．旅路の内点に接続する辺集合を E'_α とし，$E_\alpha = E_{\geq}^{i} \setminus E'_\alpha$ とする．s または z を含まない連結成分の個数分だけ旅路の内点を 1 つにまとめたダミー頂点の集合 I_α をつくる．I_α に含まれるダミー頂点を i_j ($j = \mathrm{comp}_\alpha(v)$) とする．

$V_\alpha := V[E_{\geq}^{i}] \cup \{s, z\} \cup I_\alpha \setminus \{v \in F_i \mid \mathrm{deg}_\alpha(v) = 2\}$ (13)

とする．もし，$\mathrm{comp}_\alpha(v) = 0$ を満たす v，即ち，s を含む旅路の端点 v が F_i に存在するならば，

$e_\alpha^s := \{s, v\},\ t_\alpha(e_\alpha^s) = \mathrm{time}_\alpha(v)$ (14)

とし，E_α に e_α^s を加える．z を含む旅路の端点についても同様に，$\mathrm{comp}_\alpha(v) = 1$ を満たす v が F_i に存在するならば，

$e_\alpha^z := \{z, v\},\ t_\alpha(e_\alpha^z) = \mathrm{time}_\alpha(v)$ (15)

とし，E_α に e_α^z を加える．s, z 以外のフロンティアの頂点，つまり，$\mathrm{comp}_\alpha(v) \geq 2$ の頂点に対しては，

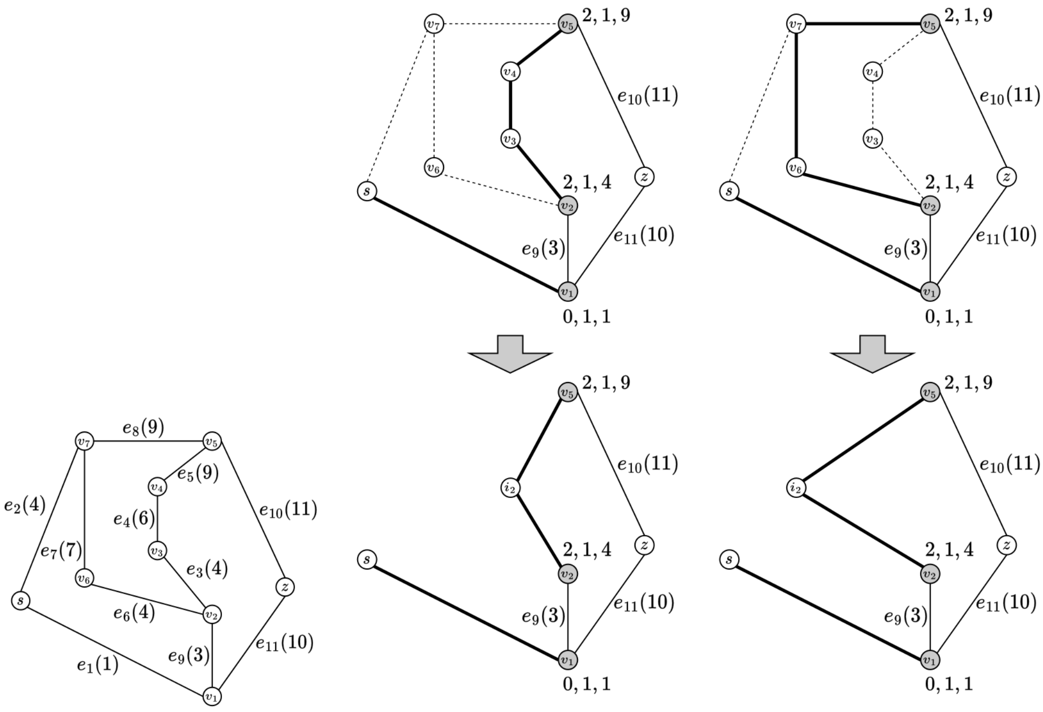
$e_\alpha^v := \{v, i_j\},\ t_\alpha(e_\alpha^v) = \mathrm{time}_\alpha(v)$ ($j = \mathrm{comp}_\alpha(v)$) (16)

とし，E_α に e_α^v を加える．以上の手続きを行うことで得られる時間的グラフ G_α を G の縮約グラフと呼ぶ．

上記の仮定から，$X \in \mathrm{E}P_\alpha$ は，$X \cup E_\alpha$ が G_α の s–z 旅路である条件を満たす．よって，(4) を満たす性質

$C_\alpha(X) \iff X \cup E_\alpha$ は G の s–z 旅路である (17)

が定義され，任意の節点対 $\beta, \beta \in N_i$ に対して，$\phi(\beta) = \phi(\beta)$ ならば，β と β は等価であるという条件を満たすことが示された．図 3 に，2 つのコンフィギュレーションが等価であるときの時間的グラフの様子を示す．2 つのコンフィギュレーションが等価であるとき，節点共有規則により，それらの節点は共有される．



(a) もとのグラフ G　　(b) α の縮約グラフ G_α　　(c) α' の縮約グラフ $G_{\alpha'}$

図 3：2 つのコンフィギュレーションが等価であるときの時間的グラフの様子．ただし，$\alpha, \alpha \in N_8$ とする．細い実線の辺を未処理の辺，太い実線の辺を採択した辺，破線の辺を採択しなかった辺とする．灰色の頂点 v_1, v_2, v_5 はフロンティアの頂点である．フロンティアの頂点の comp, deg, time の値を頂点付近に x, y, z のように示した．両者の縮約グラフにおいて，未処理の辺集合から e_9, e_{10} のみを採択した場合，s–z 旅路が完成することが共通している．このように α と α では採択した辺が異なるが，コンフィギュレーションの値が等しいため，解となるような未処理の辺の採択の仕方が一致する．また旅路となるかの判定は，コンフィギュレーションのみから定義される縮約グラフ上で可能である．

Prune(α, e_i, x) 先述の仮定に矛盾しないように枝刈りを設計する．節点 α，処理する辺 e_i，辺を採択するかを示す変数 $x \in \{0,1\}$ が与えられたとき，s から z までの旅路が存在し得ないことが確定するならば，Prune(α, e_i, x) が *True* を返すようなアルゴリズムをつくることが目的となる．以下に s–z 旅路が存在しないことが確定する条件を示す．

1. 始点 s または終点 z に 2 本以上辺が接続する
2. s と z 以外の頂点に 3 本以上辺が接続する
3. s または z の次数が 0 に確定する
4. s から z までの旅路が分断される
5. 閉路が生じる
6. 時刻ラベルの列が旅路の定義に反する

フロンティアの頂点の連結成分，次数，時刻ラベルを確認し，上記の条件のうち 1 つでも真ならば，枝刈りを実行する．図 4 に枝刈り条件 6 による枝刈りの例を示す．

枝刈り条件 1 より，フロンティアに含まれる始点および終点は，孤立点か旅路の端点のどちらかである．枝刈り条件 2，5 より，始点と終点以外のフロンティアに含まれる頂点は，孤立点，旅路の端点，旅路の内点かのいずれかである．そのため次数は 0，1，2 の 3 通りのみとなる．

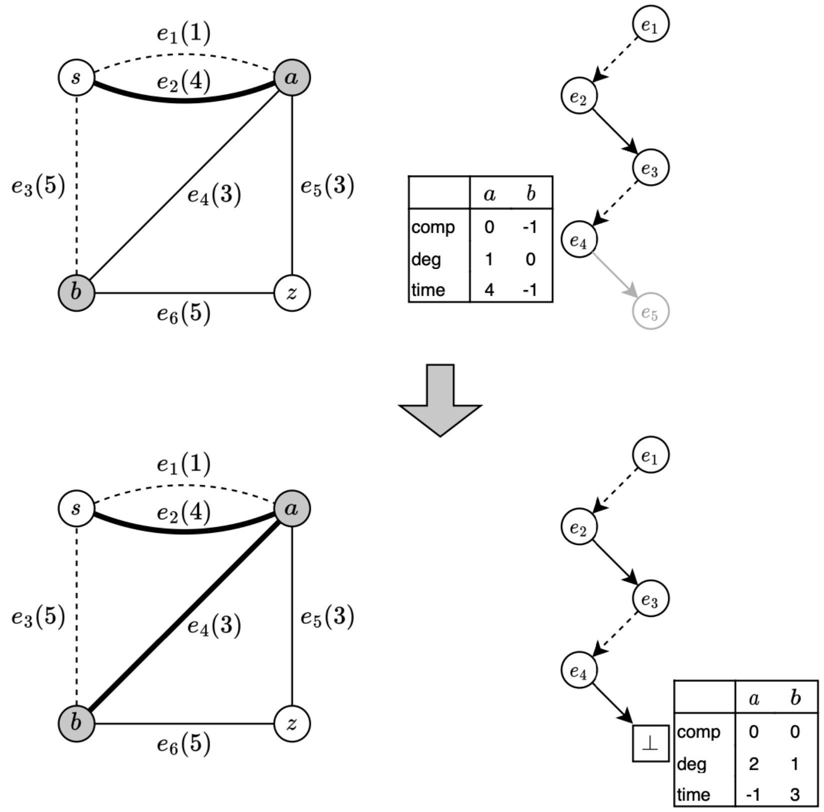


図 4: 時刻ラベルによる枝刈りの例．図左のグラフにおいて，細い実線の辺を未処理の辺，太い実線の辺を採択した辺，破線の辺を採択しなかった辺とする．灰色の頂点 a, b はフロンティアの頂点で，ラベル e_4 の節点を α とする．図の例は e_4 を採択しようとしているときの様子で，上の図がコンフィギュレーションの更新前，下の図が更新後の時間的グラフと対応する ZDD を表している．始点と連絡する旅路の端点 a の時刻ラベル $\mathrm{time}_\alpha(a)$ より $t(e_4)$ の方が小さく，採択すると旅路とならないことが確定するため，枝刈り条件 6 により枝刈りされる．

始点と終点は探索の途中でフロンティアから除かれることがあり得る．枝刈り条件 3 より，フロンティアから除かれた始点は旅路の端点である．またこのとき， $v \neq z$ とすると，フロンティアに $s \rightsquigarrow_{G_\alpha} v$ である v が必ず存在する．そうでなければ，v に接続する辺すべてが処理済みで過去にフロンティアから除かれたことになる．$v = z$ だった場合は枝刈り条件すべてに当てはまらない場合があり，v がフロンティアに存在しないことがあり得る．$v \neq z$ の場合を考える．$s \rightsquigarrow_{G_\alpha} v$ を満たす v がフロンティアに存在しないと仮定する．s, z 以外のフロンティアの頂点は孤立点か旅路の内点か旅路の端点のいずれかであるから，フロンティアから除かれた v は旅路の内点か端点のいずれかである．$v \neq z$ が旅路の端点だった場合，s–z 旅路が分断されたことになり，枝刈り条件 4 により枝刈りされているはずであり，これに矛盾する．v が旅路の内点だった場合は，$v \rightsquigarrow_{G_\alpha} v'$ となる v' が存在するが，$v' \neq z$ だった場合，先の議論から s–z 旅路が分断されたことになる． v' が内点だった場合，v' と連絡する頂点に対して同様の議論がなされる．もしフロンティアから除かれた頂点すべてが，s と連絡し，かつ，旅路の内点であるならば，旅路は閉路を成している．これは枝刈り条件 5 より枝刈りされているはずであり，これに矛盾する．よって，$v = z$ でない限り，$s \rightsquigarrow_{G_\alpha} v$ を満たす v がフロンティアに必ず含まれる．また同様の議論は終点に対しても成立し，終点 z がフロンティアから除かれた後でも，$v = s$ でない限り，$v \rightsquigarrow_{G_\alpha} z$ を満たす v が必ずフロンティアに存在する．

枝刈り条件 6 は，$s \rightsquigarrow_{G_\alpha} v$ を満たす $v \in F_i$ の，$\mathrm{time}_\alpha(v)$ が s に接続する辺の時刻ラベル以上，即ち，s から v への旅路に含まれる辺の時刻ラベルが s から v に向けて単調増加であることを保証する．終点に連絡する頂点 $w \rightsquigarrow_{G_\alpha} z$ を満たす $w \in F_i$ についても，$\mathrm{time}_\alpha(w)$ が z に接続する辺の時刻ラベル以下，即ち，

w から z への旅路に含まれる辺の時刻ラベルが w から z に向けて単調増加であることを保証する．そのため，始点または終点がフロンティアから除かれた後でも，それと連絡するフロンティアの頂点のコンフィギュレーションを参照することで，始点からの部分旅路および終点への部分旅路を構築することができる．

Algorithm 2 に枝刈りのアルゴリズムを示す．関数 CheckTimeCondition は，$e_i = \{u, v\}$ を採択したとき，u を端点にもつ旅路 J_u，v を端点にもつ旅路 J_v の和集合 $J_{new} = J_u \cup e_i \cup J_v$ について，$\mathrm{time}_\alpha(u), \mathrm{time}_\alpha(v), t(e_i)$ の大小関係を確認し，J_{new} が旅路であることの条件を満たさない場合，*False* を返し，そうでない場合は *True* を返す関数である（詳細なアルゴリズムは付録 B の Algorithm 5 に示す）．旅路がマルチホップ型かシングルホップ型かは，CheckTimeCondition 関数内部で，時刻ラベルの大小関係を表す不等号に等号をつけるか否かで簡単に対応できる．時刻ラベルの大小関係を確認する際，u と連絡する頂点がフロンティアにないか確認し，存在する場合はその頂点の時刻ラベルも確認する必要がある．v に対しても同様であるが，この手続きの計算量は $O(|F_i|)$ であるため，関数 CheckTimeCondition の計算量は $O(|F_i|)$ である．関数 UpdateInfo は，コンフィギュレーションを参照として受け取り，連結成分，次数，時刻ラベルを更新する関数であり，Algorithm 3 に示されている．関数 UpdateInfo は，$e_i = \{u, v\}$を採択するときに限り，u と v の連結成分を $\mathrm{comp}_\alpha(u)$ と $\mathrm{comp}_\alpha(v)$ の値の小さい方に揃え，$\mathrm{deg}_\alpha(u) \leftarrow \mathrm{deg}_\alpha(u)+1, \mathrm{deg}_\alpha(v) \leftarrow \mathrm{deg}_\alpha(v)+1$ と次数を更新する．更新後の次数が 1 ならば，$\mathrm{time}_\alpha(u) \leftarrow t(e_i), \mathrm{time}_\alpha(v) \leftarrow t(e_i)$ と時刻ラベルを更新する．

更新後の次数が 2 ならば，$\mathrm{time}_\alpha(u) \leftarrow -1, \mathrm{time}_\alpha(v) \leftarrow -1$ と更新する．連結成分の更新に $O(|F_i|)$，次数，時刻ラベルの更新に $O(1)$ かかるため，関数 UpdateInfo の計算量は $O(|F_i|)$ である．

GenerateNode(α, e_i, x) GenerateNode 関数の主たる役割は，コンフィギュレーションの更新を行うことである．コンフィギュレーションの更新は UpdateInfo 関数と同様の手続きをとるため，これを流用する．UpdateInfo 関数と異なるのは，$i+1$ の変数ラベルをもつ節点の準備を行う点である．まず $i+1$ の変数ラベルをもつ節点 β を生成し，$\phi(\alpha)$ を $\phi(\beta)$ にコピーする．次に，以下の手順でコンフィギュレーションを更新する．

1. $x = 0$ ならば，更新しない．$x = 1$ ならば，UpdateInfo$(\beta, e_i, 1)$ を実行
2. フロンティアから除かれる F_i ¥ F_{i+1} の頂点に対応する列を削除し，新たにフロンティアに加わる F_{i+1} / F_i の頂点に対応する列を挿入し初期化（新たにフロンティアに加わる頂点 $v \in F_{i+1} / F_i$ のコンフィギュレーションに対して，$\mathrm{comp}_\beta(v) \leftarrow -1, \mathrm{deg}_\beta(v) \leftarrow 0, \mathrm{time}_\beta(v) \leftarrow -1$ の値を代入）

$|F_i/F_{i+1}|, |F_{i+1}/F_i|$はいずれも 0 か 1 か 2 のため，削除される列の数は高々定数個である．節点のコピーの計算量は $O(|F_i|)$で，UpdateInfo 関数の計算量は $O(|F_i|)$ であるため，GenerateNode 関数の計算量は $O(|F_i|)$ である．

3-3 旅路列挙のためのフロンティア法の計算量

旅路列挙のためのフロンティア法の計算量を求める．i 番目の節点数は $|N_i|$ に等しい．$|N_i|$は高々，ラベル e_i の節点に対して，行列 ϕ のとりうるパターン数である．フロンティアの頂点 v の次数は 0, 1, 2 の 3 通りで，連結成分は $|F_i|$ 通りで，時刻ラベルは T 通りであり，フロンティアの頂点数が $|Fi|$ であるから，$|Ni|$

は高々$(3 \cdot |F_i| \cdot T)^{|Fi|}$ である．枝刈りの計算量は $O(|F_i|)$，受理の計算量は $O(1)$，GenerateNode の計算量は $O(|F_i|)$であるから，旅路列挙のフロンティア法の計算量は

$$O\left(\sum_{i=1}^{m} |F_i| \cdot (|F_i|T)^{|F_i|}\right) \quad (18)$$

である．旅路列挙のフロンティア法は入力される時間的グラフの頂点数の影響を直接受けない点が特徴であり，比較的頂点数の大きな時間的グラフに対しても旅路列挙ができることを示唆する．また計算量はフロンティアサイズに対して指数的に増加する．フロンティアサイズは処理する辺の順序に依存するため，処理する辺の順序をフロンティアサイズの最大値が最小化するように並べることは，計算時間を削減することに寄与する．フロンティアサイズの最大値を最小化する辺順序を求める問題は一般には NP 困難で，この問題を近似的に解く手法が提案されている [18]．辺順序に関しては 4.1 節に述べる．

次の 4 章では，旅路 ZDD の構築，superset 演算，信頼性計算の各ステップでどのくらい時間がかかるか，実験により評価する．

4 実験

提案手法の有効性を確かめるために計算機実験を行った．提案手法と，既存手法の旅路を列挙する部分は C++（g++8.5.0 with the -O3 option）で実装した．提案手法の実装に際しては，高度に最適化された実装がなされたフロンティア法のフレームワークである,TdZdd ライブラリ(https://github.com/kunisura/ TdZdd) を利用した.既存手法の信頼性計算の部分は,Chaturvedi らが公開している SDP 法のプログラムを利用した．ただし，SDP 法のプログラムだけは MATLAB で実装されており，辺数が 53 以下のインスタンスしか扱えないという制約があることに注意する．また，superset 演算のアルゴリズムに関しては，使用したライブラリに ZDD を出力するものがあり，独自に実装した BDD を出力するものと計算時間を比較することとした．以後，便利のため，superset 演算後に BDD を出力する提案手法を手法 B，ZDD を出力するものを手法 Z と呼ぶ．マシンスペックは以下の通りである．

- CPU: Intel Xeon Gold 6238L (22 cores × 4)
- Memory: 512 GB (allocated by slurm job scheduler)
- OS: RedHat Enterprise Linux 8.5

4-1 データセット

実験対象とした時間的グラフのインスタンスは，Chaturvedi らの論文 [11]を参考にし，静的グラフの各辺にある確率で時刻ラベルを付与することで作成した．静的グラフの各辺に割り当てる時刻ラベルは 1 以上 T 以下の整数とする．静的グラフの辺に時刻ラベルを付与することで，時間的グラフの辺が生成されるが，ここでは，静的グラフの辺に時刻ラベルが付与される確率を一律 0.5 とする．時刻ラベルの付与は確率的に行われるため，時刻ラベルがまったく付与されない場合がある．そのため，時間的グラフのインスタンスの作成は，始点または終点を含む辺が時間的グラフに含まれるまで，繰り返し行うこととする．なお，各辺の生存確率は一律 0.9 とした．対象とした静的グラフは完全グラフと高さ 3 のグリッドグラフとした．完全グラフとグリッドグラフについては，フロンティアサイズの最大値が最小となるような幅優先の辺順序を用いた．完全グラフは，頂点数 n が 3 から 10 までのものを用意し，$T = n - 1$ とした．高さ 3 のグリッドグラフは，幅 w が 3 から 10 のものを用意し，$T = 2w$ とした．n 頂点の完全グラフについては，その頂点集合を $V = \{v_1, ..., v_n\}$ とするとき，始点 s を v_1，終点 z を v_n とした．高さ h，幅 w のグリッドグラフについては，その頂点集合を $V = \{v_1, ..., v_{hw}\}$ とするとき，s と z がグリッドグラフの対角線上に位置するよう，s は v_1，z は v_{hw} とした．平均のプログラム計算時間を計測するために，完全グラフの時間的グラフのインスタンスは，各 n に対して 100 個用意した．グリッドグラフのインスタンスも同様に，各 w に対して 100 個用意した．プログラムの実行は，完全グラフの場合は各 n の場合の 100 個のインスタンスを入力として行うが，n の i 番目のインスタンスを入力としたときのプログラムの計算時間が 2 時間を超えた場合，プログラムの実行を中止し，n の i 番目以上のインスタンスと $n + 1$ 以上のすべてのインスタンスに対する実験を中止した．グリッドグラフの場合も同様に w の i 番目のインスタンスを入力としたときのプログラムの計算時間が 2 時間を超えた場合，w の i 番目以上のインスタンスと $w + 1$ 以上のインスタンスすべてに対する実験を中止した．

4-2 既存手法との比較

既存手法の SDP 法の部分の実装は Chaturvedi らが公開している MATLAB のプログラムを利用したが，このプログラムは 53 個以下の辺のグラフしか取り扱えないことを先に述べた．そのため，既存手法との比較実験は，平均の辺数が 53 以下となる，頂点数 n が 3 から 6 までの完全グラフの時間的グラフを対象とした．表 1 に計算時間を示す．n の列は完全グラフの頂点数を表し，$m_{\mathrm{avg.}}$ の列は平均の辺数を表す．$t_{\mathrm{extg.}}$，t_{B}, t_{Z} の列はそれぞれ，既存手法の平均の計算時間，手法 B の平均の計算時間，手法 Z の平均の計算時間を表す．計算時間の単位はいずれも秒である．$t_{\mathrm{extg.}}/t_{\mathrm{B}}$ の列は，既存手法の計算時間を手法 B の計算時間で割った値を表し，既存手法と比べ，手法 B が何倍高速かを意味する．$t_{\mathrm{extg.}}/t_{\mathrm{Z}}$ の列も同様に，既存手法と比べ，手法 Z が何倍高速かを意味する．

すべての頂点数の場合で，提案手法は既存手法と比べ平均の計算時間が小さい．マルチホップの場合，手法 B は最大約 37,372 倍，手法 Z は最大約 8,309 倍高速であった．シングルホップの場合，手法 B は最大約 814 倍，手法 Z は最大約 774 倍高速であった．よって，既存手法と比べ，提案手法は TVN 信頼性計算をより効率よく計算できるものと考えられる．また，手法 B は手法 Z より高速である傾向がある．この理由は後節で詳しく説明するが，superset 演算と信頼性計算の計算時間が手法 Z より小さいからである．

表 1: 完全グラフに対する実験結果と計算時間. $m_{\mathrm{avg.}}$ を平均の辺数, $t_{\mathrm{extg.}}$ を既存手法の平均の計算時間, t_{B} を手法 B の平均の計算時間, t_{Z} を手法 Z の平均の計算時間とする.

(a) マルチホップの場合

n	$m_{\mathrm{avg.}}$	$t_{\mathrm{extg.}}$ (s)	t_{B} (s)	$t_{\mathrm{extg.}}/t_{\mathrm{B}}$	t_{Z} (s)	$t_{\mathrm{extg.}}/t_{\mathrm{Z}}$
3	3	1.13×10^{-1}	4.49×10^{-4}	252	4.85×10^{-4}	233
4	9	3.16×10^{-1}	4.96×10^{-4}	637	5.45×10^{-4}	580
5	20	7.45×10^{-1}	6.75×10^{-4}	1105	8.80×10^{-4}	847
6	37.5	1.50×10^{2}	4.02×10^{-3}	37 372	1.81×10^{-2}	8309

(b) シングルホップの場合

n	$m_{\mathrm{avg.}}$	$t_{\mathrm{extg.}}$ (s)	t_{B} (s)	$t_{\mathrm{extg.}}/t_{\mathrm{B}}$	t_{Z} (s)	$t_{\mathrm{extg.}}/t_{\mathrm{Z}}$
3	3	1.30×10^{-1}	4.64×10^{-4}	279	4.93×10^{-4}	263
4	9	4.11×10^{-1}	5.04×10^{-4}	814	5.31×10^{-4}	774
5	20	3.32×10^{-1}	6.11×10^{-4}	544	6.69×10^{-4}	497
6	37.5	1.02	1.28×10^{-3}	800	1.99×10^{-3}	514

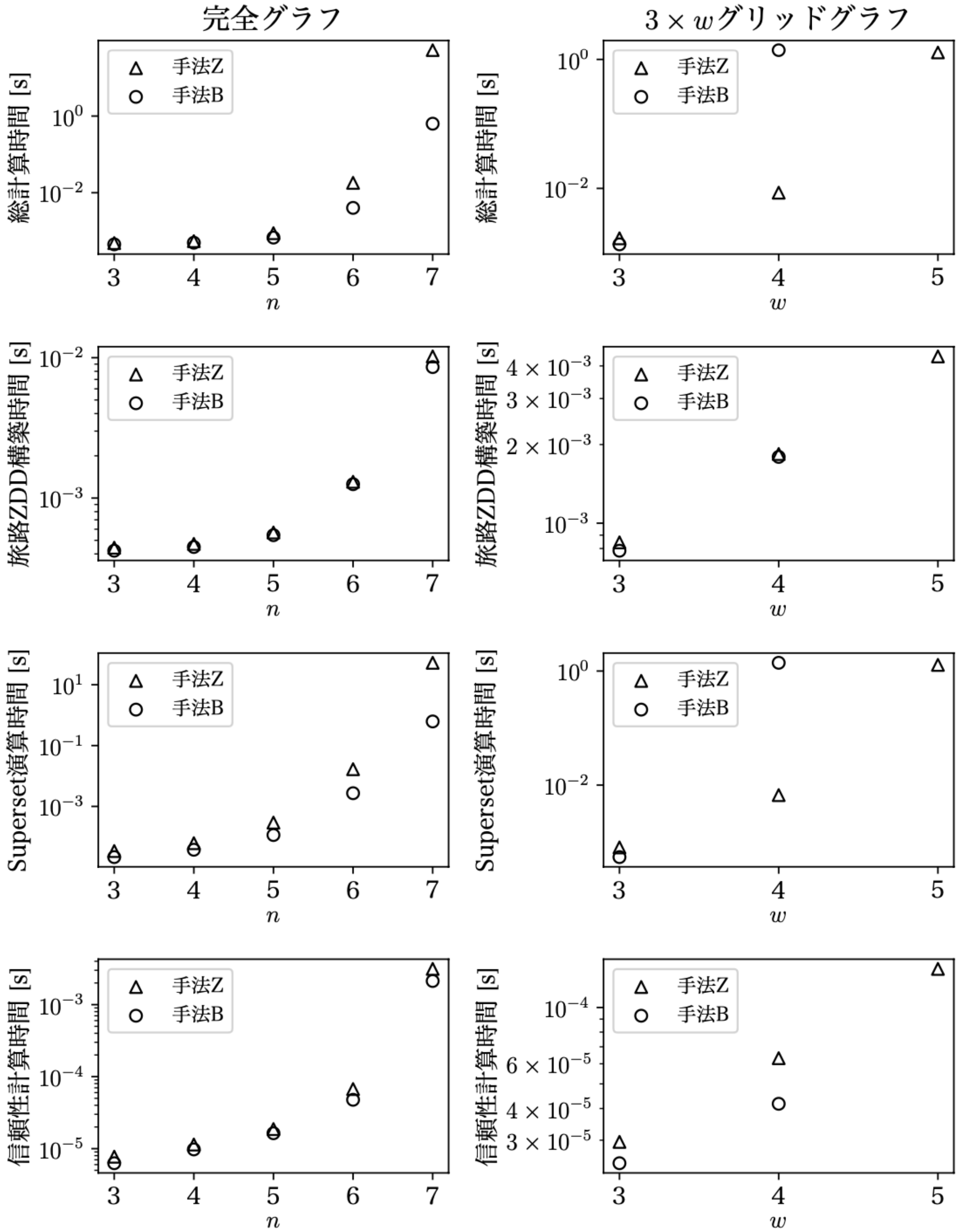
4-3 手法 B と手法 Z の比較

全体の計算時間だけでなく，各ステップの計算時間を評価するため，図 5 に手法 B と手法 Z の各計算時間の散布図を示す．図 5(a)はマルチホップの場合のもので，図 5(b)はシングルホップの場合のものである．左列の散布図は完全グラフの場合のもので，縦軸に計算時間，横軸に頂点数 n をとったものである．右列の散布図はグリッドグラフの場合のもので，縦軸に計算時間，横軸にグリッドグラフの幅 w をとったものである．1 段目の散布図は全体の計算時間をプロットしたもので，2 段目は旅路 ZDD の構築部分の計算時間，3 段目は superset 演算の計算時間，4 段目は動的計画法による信頼性の計算時間をプロットしたものである．

完全グラフに対しては，手法 B が手法 Z より高速だが，グリッドグラフに対しては，手法 Z がより高速な傾向にある．これは，superset 演算の計算時間の違いが原因だと考えられる．完全グラフの場合，手法 B の superset 演算と信頼性計算の計算時間が手法 Z と比べ小さく，これが手法 B がより高速となった理由だと考えられる．一方，グリッドグラフの場合，手法 B の信頼性計算の計算時間は手法 Z より小さいが，superset 演算の計算時間が手法 Z より大きく，完全グラフのときと異なり，手法 Z がより高速となったと考えられる．完全グラフとグリッドグラフで superset 演算の計算時間の傾向が異なった原因として，辺密度の影響と実装方法の影響の 2 つが考えられるが，実装方法の影響が主な原因と考えられる．手法 Z の superset 演算は予め用意されていたライブラリの関数を利用したため，高度に最適化がなされているが，手法 B の superset 演算は既存手法［17］を参考に独自に実装したため，このような傾向となったと考えられる．また，旅路 ZDD の構築にかかった平均時間と動的計画法による信頼性計算の平均計算時間は 0.1 秒以下であるが，superset 演算にかかった平均時間は最大で約 57 秒であることから，superset 演算がボトルネックとなっていると考えられる．

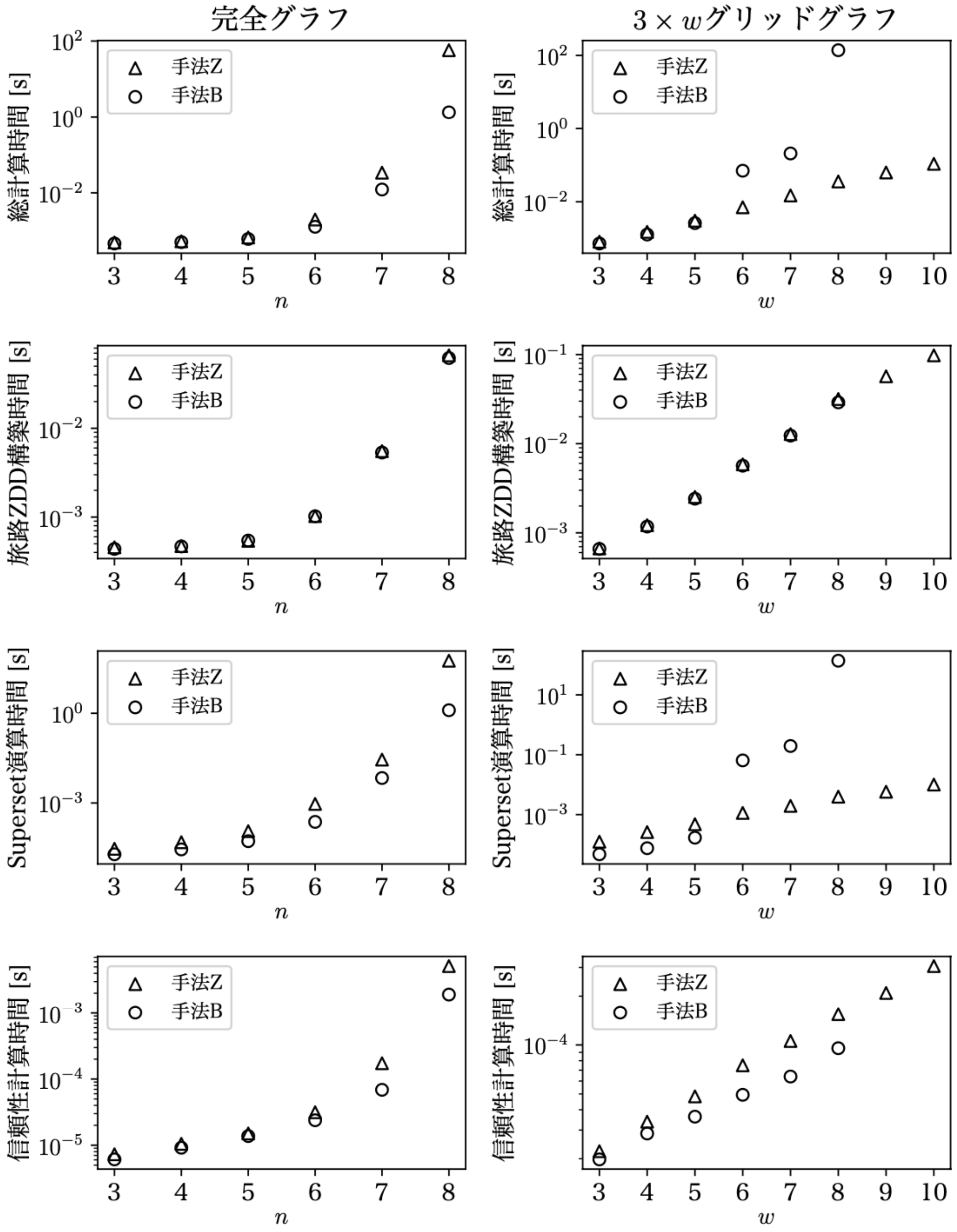
図 5 から，信頼性計算の計算時間に手法による違いがあることがわかる．この理由を説明するため，図 6 に手法 B と手法 Z の始終点連絡辺集合の集合族を表す二分決定グラフの節点数の散布図を示す．信頼性計算の計算量は節点数に線形比例するため，節点数を削減することで計算時間を削減することができる．図 6(a)より，マルチホップの場合，完全グラフでは，手法 B の節点数は手法 Z の節点数の約 6 割から約 7 割程度に抑えられており，グリッドグラフでは，約 4 割程度に抑えられていることがわかる．シングルホップの場合，図 6(b)より，完全グラフでは，手法 B の節点数は手法 Z の節点数の約 3 割から約 5 割程度に抑えられており，グリッドグラフでは，約 1 割から約 2 割程度に抑えられていることがわかる．そのため，いずれの場合も BDD を出力した方が節点数を抑えられ，その効果はグリッドグラフの方が顕著に現れると考えられる．

また図 5の結果と併せて，節点数を削減することで信頼性計算時間を削減でき，またメモリの使用量も削減できると考えられるため，手法 B のアプローチがより好ましいと考えられる．



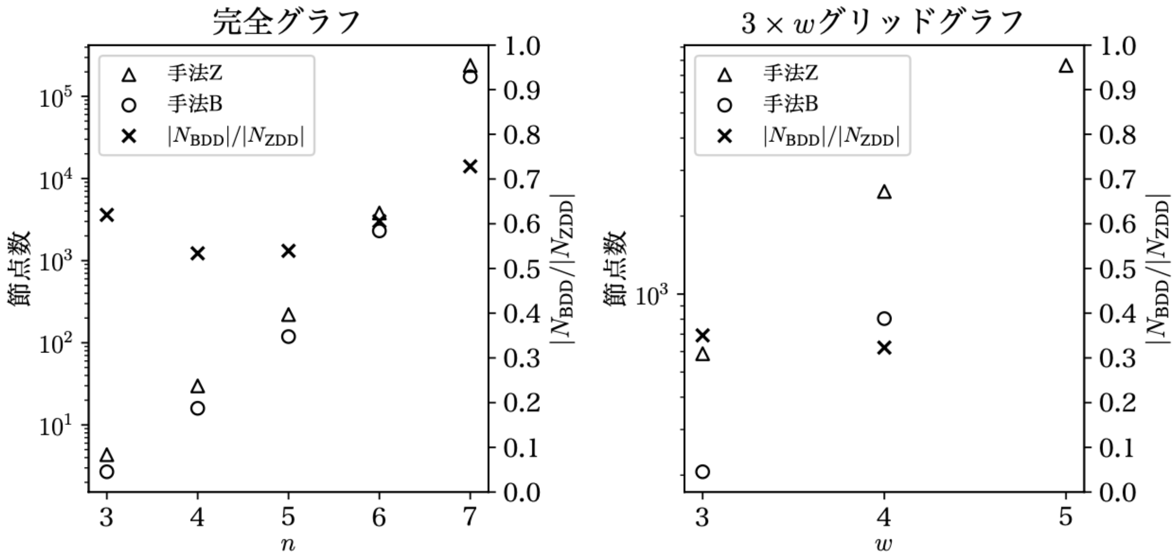
(a) マルチホップの場合

公益財団法人電気通信普及財団
研究調査助成報告書　第38号　2023年度

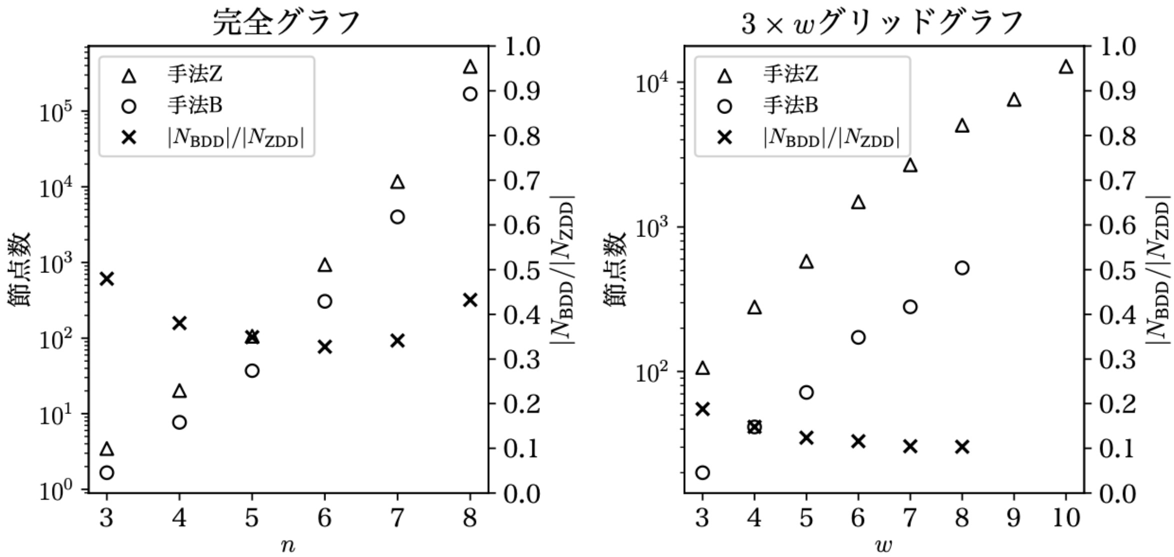


(b) シングルホップの場合

図 5: 手法 B と手法 Z の計算時間の比較. 結果が散布図にない実験は 2 時間の計算時間の制限により中止された.



(a) マルチホップの場合



(b) シングルホップの場合

図 6: 手法 B と手法 Z の節点数の比較. N_{BDD} を BDD の節点数, N_{ZDD} を ZDD の節点とする.

【参考文献】

[1] Hebert P´erez-Ros´es. "Sixty Years of Network Reliability". Math. Comput. Sci., 12(3):275–293, 2018.

[2] E.F. Moore and C.E. Shannon. Reliable Circuits using Less Reliable Relays.
Journal of the Franklin Institute, 262(3):191–208, September 1956.

[3] Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. SIAM J. Comput., 8(3):410–421, 1979.

[4] J. Scott Provan. The Complexity of Reliability Computations in Planar and Acyclic Graphs. SIAM J. Comput., 15(3):694–702, 1986.

[5] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans. Computers, 35(8):677–691, 1986.
[6] Hiroshi Imai. Computational Investigations of All-Terminal Network Reliability via BDDs. IEICE Trans. Fundamentals, 82(5):714–721, 1999.
[7] Gary Hardy, Corinne Lucet, and Nikolaos Limnios. K-Terminal Network Reliability Measures With Binary Decision Diagrams. IEEE Trans. Reliab., 56(3):506–515, 2007.
[8] Chamitha de Alwis, Anshuman Kalla, Quoc-Viet Pham, Pardeep Kumar, Kapal Dev, Won-Joo Hwang, and Madhusanka Liyanage. Survey on 6G Frontiers: Trends, Applications, Requirements, Technologies and Future Research. IEEE Open J. Commun. Soc., 2:836–886, 2021.
[9] Othon Michail. An Introduction to Temporal Graphs: An Algorithmic Perspective. Internet Math., 12(4):239–280, 2016.
[10] Sanjay Kumar Chaturvedi. Network Reliability: Measures and Evaluation.
John Wiley & Sons, 2016.
[11] Sanjay Kumar Chaturvedi, Gaurav Khanna, and Sieteng Soh. Reliability Evaluation of Time Evolving Delay Tolerant Networks Based on Sum-ofDisjoint Products. Reliab. Eng. Syst. Saf., 171:136–151, 2018.
[12] Sandeep Bhadra and Afonso Ferreira. Complexity of Connected Components in Evolving Graphs and the Computation of Multicast Trees in Dynamic Networks. In Ad-Hoc, Mobile, and Wireless Networks, Second International Conference, ADHOC-NOW 2003 Montreal, Canada, October 8-10, 2003, Proceedings, pages 259–270, 2003.
[13] Shin-ichi Minato. Zero-suppressed BDDs and Their Applications. Int. J. Softw. Tools Technol. Transf., 3(2):156–170, 2001.
[14] Shin-ichi Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In Proceedings of the 30th Design Automation Conference.
Dallas, Texas, USA, June 14-18, 1993., pages 272–277, 1993.
[15] Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin-ichi Minato.
Frontier-Based Search for Enumerating All Constrained Subgraphs with Compressed Representation. IEICE Trans. Fundam. Electron. Commun. Comput. Sci., 100-A(9):1773–1784, 2017.
[16] Donald E. Knuth. The Art of Computer Programming, Volume 4A: Combinatorial Algorithms Part 1.
Addison-Wesley, 2011.
[17] Takahisa Toda, Shogo Takeuchi, Koji Tsuda, and Shin-ichi Minato. Superset Generation on Decision Diagrams. In WALCOM: Algorithms and Computation - 9th International Workshop, WALCOM 2015, Dhaka, Bangladesh, February 26-28, 2015. Proceedings, pages 317-322, 2015.
[18] Yuma Inoue and Shin-ichi Minato. Acceleration of ZDD Construction for Subgraph Enumeration via Path-width Optimization. TCS-TR-A-16-80.
Hokkaido University, 2016.
[19] Lutz Oettershagen, Petra Mutzel, and Nils M. Kriege. Temporal Walk Centrality: Ranking Nodes in Evolving Networks. In WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022, pages 1640–1650, 2022.
[20] Takanori Maehara, Hirofumi Suzuki, and Masakazu Ishihata. Exact Computation of Influence Spread by Binary Decision Diagrams. In Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017, pages 947–956, 2017.
[21] Hirofumi Suzuki, Masakazu Ishihata, and Shin-ichi Minato. Exact Computation of Strongly Connected Reliability by Binary Decision Diagrams. In Combinatorial Optimization and Applications - 12th International Conference, COCOA 2018, Atlanta, GA, USA, December 15-17, 2018, Proceedings, pages 281–295, 2018.

〈発表資料〉

題名	掲載誌・学会名等	発表年月
時間変化するネットワークに対する二分決定グラフを用いた信頼性評価法	2022 年度オペレーションズ・リサーチ学会関西支部若手研究発表会	2022 年 10 月