Vol.2017-AL-165 No.16
2017/11/17

BDDを用いたリンクの故障に依存関係がある場合のネットワーク信頼性評価

吉田 拓弥[1]　川原 純[1]　井上 武[2]　笠原 正治[1]

概要：ネットワーク信頼性評価とは，ネットワークの各リンクに静的な故障確率が設定されている場合に，2 頂点間が通信可能である確率を求める問題である．確率を厳密に計算する手法として，二分決定グラフ (BDD) を用いた計算方法が知られている．BDD は論理関数を圧縮して効率よく表現できるデータ構造である．本稿では，ネットワークの 2 つ以上のリンクの故障に依存関係がある場合の信頼性評価を行う．本手法では，リンク間に存在する依存関係を BDD で表現し，依存関係を考慮しない場合に構築した BDD との二項演算を行うことで，依存関係を考慮した信頼性 BDD を生成し，確率を計算する．本手法を 3 つの計算方法で実装し，各方法を処理時間と生成される BDD のノード数の観点から比較を行う．

キーワード：グラフアルゴリズム，二分決定グラフ，ネットワーク信頼性

On network reliability with link failure dependencies using BDDs

Yoshida Takuya[1]　Jun Kawahara[1]　Takeru Inoue[2]　Shoji Kasahara[1]

Abstract: The network reliability evaluation is to find the probability that two specified nodes can communicate with each other in a given network whose links break down with some probabilities. Calculation methods based on Binary Decision Diagram (BDD) are known as strict methods for the network reliability evaluation. A BDD is a data structure that can efficiently express Boolean functions. In this research, we propose methods to calculate the reliability considering dependencies between links of a network. We implement three calculation methods and compare them in terms of the running time and BDD size.

Keywords: Graph algorithm, Binary decision diagram, Network reliability

1. 研究背景

ネットワークの各リンク（以下，辺と呼ぶ）に対し，静的かつ独立に故障する確率が与えられた際，任意の 2 点間が通信可能である確率の計算をネットワーク信頼性評価 (network reliability evaluation) と呼ぶ．ネットワーク信頼性評価は古くから研究されている基礎的な問題である [1], [2]．ネットワーク信頼性評価の確率を厳密に計算するための時間計算量は，Valiant [3] により#P 完全であることが証明されている．本問題が#P 完全に属することは，即ち大規模ネットワークに対して，効率の良い確率計算アルゴリズムの設計が困難であることを示している．

その中で厳密解を計算する有効な手法として，グラフの辺を削除・縮約 (edge deletion-contraction) することにより二分決定グラフ (Binary Decesion Diagram; BDD) [4], [5] を構築する手法が提案されている [6]．ネットワークの各リンクが故障している場合とそうでない場合に分けて二分木を構築し，途中状態が同一なものを併合することにより，重複する箇所の計算を省略できる．これにより，計算時間及び使用メモリ量を大幅に削減できる．構築した BDD を信頼性 BDD と呼ぶ．この手法を用いて，100 頂点規模のネットワークに対し厳密解の求解に成功している [7]．また，ネットワークの辺だけでなく，ノード（以下，頂点と呼ぶ）が故障する場合のネットワーク信頼性評価 [8], [9] 等

[1] 奈良先端科学技術大学院大学 情報科学研究科
[2] NTT 未来ねっと研究所

も提案されてきた．

本稿では，ネットワークの 2 つ以上の辺が同時に壊れる場合について，ネットワーク信頼性評価を行う手法を提案する．本稿では，2 本以上の辺について，1 本の辺が壊れているならば，残りの辺も壊れており，また，1 本の辺が壊れていないならば，残りの辺も壊れていないとき，それらは依存関係を持つと呼ぶ．

提案手法では，依存関係を満たす論理関数を BDD で表現し，それらの BDD と信頼性 BDD の二項演算を行い，依存関係を満たす信頼性 BDD を構築する．BDD の演算順は複数考えられるので，本稿では 3 つの方法について数値実験により比較を行う．

2. 準備

本章では，まず取り扱う問題の定式化を行い，BDD の性質と構築法，BDD を用いたネットワーク信頼性評価の計算方法についてそれぞれ述べる．

2.1 問題の定式化

頂点集合 $V = \{v_1, v_2, \ldots, v_n\}$，辺集合 $E = \{e_1, e_2, \ldots, e_m\}$ から構成される連結グラフ $G = (V, E)$ において，辺にそれぞれ故障確率 p_i $(i = 1, 2, \ldots, m)$ が与えられるとする．辺 e_i は p_i の確率で故障状態であり，$1 - p_i$ の状態で接続状態である．このとき，ある 2 頂点 a, b $(\in V)$ 間に接続状態の辺でのみ構成されるパスが 1 つでもあれば，その 2 点間は通信可能であり，パスが 1 つも存在しなければ通信不可能であるとする．本稿では，グラフ G，故障確率 p_i，グラフ中の 2 点 a, b が与えられたとき，2 点間が通信可能である確率を求める問題を，（依存関係を考慮しない）ネットワーク信頼性評価と呼ぶ．

次に，本稿で扱うネットワークの辺の依存関係について述べる．本稿で扱う依存関係は，E の部分集合 $E' = \{e_{f(1)}, e_{f(2)}, \ldots, e_{f(\ell)}\}$ $(f(1) < f(2) < \cdots < f(\ell))$ の形として与えられる．依存関係 E' とは，E' のすべての辺が故障しているか，すべての辺が故障していないか，どちらかの状態を取ることを意味する．このとき，E' のすべての辺が故障している確率は $p_{f(1)}$ であり，E' のすべての辺が故障していない確率は $1 - p_{f(1)}$ であるとする．$p_{f(2)}, \ldots, p_{f(\ell)}$ の値は無視する．$\mathcal{E}$ を依存関係の集合とする．依存関係を考慮したネットワーク信頼性評価とは，$\mathcal{E}$ のすべての依存関係を満たすときの，2 点間が通信可能である確率を求める問題である．

2.2 BDD

BDD [4], [5] は，論理関数を非巡回有向グラフにより表現したものである．BDD はサイズの大きい論理関数をコンパクトかつ一意に表現することができるという特徴を持つ．BDD の例を図 1 に示す．図 1 の BDD は論理関数 $(a \wedge b) \vee (a \wedge \bar{b} \wedge \bar{c}) \vee (\bar{a} \wedge \bar{c})$ を表す．

BDD は二分決定木 (binary decision tree) を縮約することにより得られる．二分木の各ノードから出る 2 つの枝には，0 または 1 が割り当てられ (以下，0-枝，1-枝と呼ぶ)，そのノードに割り当てられた変数のうちどちらを選択するかを示す．二分木の葉にも 0 もしくは 1 の値が割り当てられる (以下，0-終端，1-終端と呼ぶ)．全ての変数割当が終了した時点で，論理関数の出力が 1 の場合は 1-終端，そうでない場合は 0-終端となる．例えば，図 1 の BDD において，上部の a のノードから順に，1-枝，0-枝，0-枝の順にたどることで，1-終端に到達するが，これは論理関数 $(a \wedge b) \vee (a \wedge \bar{b} \wedge \bar{c}) \vee (\bar{a} \wedge \bar{c})$ の変数に $a = 1, b = 0, c = 0$ を割当てたときの出力が 1 となることを意味する．枝をたどる際に現れなかった変数は don't care，すなわち 0 と 1 のどちらを割当ててもよいことを意味する．

この二分決定木の変数を場合分けする順序が常に同じになるように固定し，以下の 2 種類の縮約処理を可能な限り行うことで，既約な形の BDD が得られる．この操作の概要図を図 2 に示す．

- 0-枝と 1-枝の分岐先が同じ場合は，そのノードを削除し，削除したノードに接続していた枝を分岐先ノードにまとめる (冗長なノードの削除)．
- 場合分けする変数が同じで，0-枝と 1-枝が指すノードが同じノードが 2 個以上存在する場合，ノードを 1 つにまとめて他ノードを削除した後，削除したノードを指していた枝の先をまとめたノードに変更して等価な途中経過として記憶（等価なノードの共有）．

BDD の応用例として，グラフ列挙問題への適用が挙げられる．グラフ $G = (V, E)$，頂点集合 $V = \{v_1, v_2, \ldots, v_n\}$，辺集合 $E = \{e_1, e_2, \ldots, e_m\}$ を考えると，グラフ列挙問題

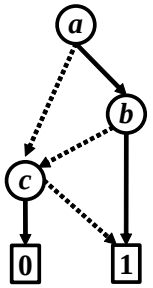

図 1 BDD の例．論理関数 $(a \wedge b) \vee (a \wedge \bar{b} \wedge \bar{c}) \vee (\bar{a} \wedge \bar{c})$ を表す．丸がノード，四角が終端記号，点線が 0-枝，実線が 1-枝を表す．

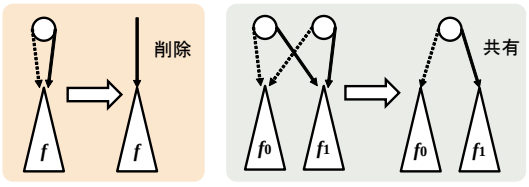

図 2 既約な形の BDD を得るための操作

は辺の冪集合 2^E からある条件を満たす集合を求める問題であると言える．グラフ列挙問題の解集合を辺の組合せ集合と考え，グラフの辺を BDD のノード変数とすれば，BDD でグラフの列挙問題を取り扱うことが可能となる．

2.3 信頼性 BDD 構築法

BDD を構築する上で，二分決定木を生成してからこれを圧縮するという手法は，変数の数に対して指数関数的な時間と記憶領域が必要となるため，現実的な構築手段ではない．辺の削除・縮約法 [6] と呼ばれる BDD 構築アルゴリズムは，二分決定木を介さずに，BDD 生成途中で等価なノードの共有を可能な限り行う手法である．この手法を，信頼性評価だけでなく，パスやマッチング等の様々な部分グラフ列挙に汎用化した手法がフロンティア法 [10] である．フロンティア法はノードの生成と並行し，ノードの共有を可能な限り行う．これにより，生成するノードの数を削減することで，計算量を改善させている．本節では，フロンティア法の概要について述べる．

図 3 にフロンティア法によるノードの等価判定についての例を示す．図 3 はあるネットワークに対し，同じ辺まで処理した途中状態を示している．楕円の左側に描かれている実線が既に処理済みの辺であり，楕円の右側に描かれている点線が未処理の辺である．左側に描かれている実線のうち，太線が変数として選ばれた辺，細い線が選ばれなかった辺である．左右 2 つのグラフをそれぞれ比較すると，変数の選び方は異なるものの，いずれも楕円中の頂点 a は頂点 s と接続しており，頂点 c はいずれの頂点とも接続していない．

このように，楕円内にある全頂点について，楕円の左側の辺の接続状況が同じであれば，途中経過の段階での通信可能判定が同じになる．この楕円のように，処理済みの辺と未処理の辺の両方に接続している頂点集合をフロンティアと呼ぶ．フロンティア法ではこの頂点集合の接続状況のみを記憶して BDD ノードの等価判定を行う．BDD 構築の詳細は文献 [6], [10] を参照されたい．

2.4 ネットワーク信頼性評価の計算方法

フロンティア法によって構築した信頼性 BDD を用いて連結確率を計算する．BDD が表現する各ネットワーク状態について実現確率を求め，その総和を取ることでネットワーク信頼性評価を行う．動的計画法を用いて計算を効率化する．

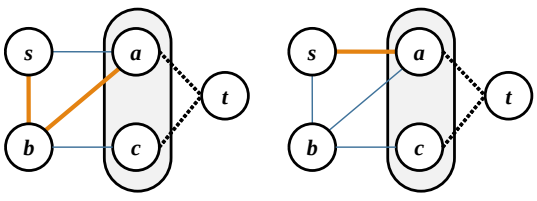

図 3 フロンティアの例

信頼性 BDD の根ノードから幅優先探索を行い，各ネットワーク状態について実現確率を求めていく．根ノードの実現確率を 1 とすると，i 段目の各ノードから伸びる 1-枝から繋がるネットワーク状態の実現確率は，i 段目のノードの実現確率に e_i の接続確率 $1-p_i$ を掛け合わせたものになる．同様に，0-枝から伸びるネットワークの実現確率は，i 段目のノードの実現確率に e_i の故障確率 p_i を掛け合わせたものになる．両者を足し合わせた値がそのノードの実現確率である．この計算を m 段目まで行うことで，1-終端記号に 1-終端記号に 2 頂点 a，b が通信可能である確率が出力される．この 1-終端記号の生成確率の総和を取ることで，ネットワークの信頼性確率が求まる．

3. 依存関係を考慮したネットワーク信頼性評価アルゴリズム

本節では，ネットワークのリンク間の依存関係を考慮したネットワーク信頼性評価アルゴリズムについて述べる．

3.1 提案手法

本項では，依存関係を考慮したネットワーク信頼性評価を行う手法について述べる．アルゴリズムの概要を以下に示す．

(1) 依存関係を考慮しないネットワーク信頼性 BDD を生成する
(2) 依存関係を表現する BDD を生成する
(3) (1) と (2) で生成した 2 つの BDD の論理積を取る

上のアルゴリズム (2) で用いる，依存関係を BDD で表現する方法について述べる．ある依存関係 $E' \in \mathcal{E}$ について，$E' = \{e'_1, e'_2, \ldots, e'_\ell\}$ とする．変数の順は $e'_1, e'_2, \ldots, e'_\ell$ の順であると仮定する．このとき，$e'_1, e'_2, \ldots, e'_\ell$ に対応する論理変数をそれぞれ $x_{e'_1}, x_{e'_2}, \ldots, x_{e'_\ell}$ とする．依存関係 E' を満たす場合に 1 を，それ以外の場合に 0 を出力する論理関数 $F_{E'}$ は，

$$F_{E'} = \bigwedge_i x_{e'_i} \vee \bigwedge_i \bar{x}_{e'_i}$$

と表せる．

$F_{E'}$ を表す BDD の構築は容易である．初めにノード e'_1（ラベル e'_1 が付与されたノード）を作成する．$x = 0, 1$ に対して以下の操作を行う．e'_1 の x-枝の先にノード e'_2 を作成する．e'_2 の $(1-x)$-枝（すなわち x-枝のもう一方の枝）の先は 0-終端となる．以降，同様に，e'_i の $(1-x)$-枝の先は 0-終端とし，x-枝の先はノード e'_{i+1} とする．ノード e'_ℓ の x-枝の先は 1-終端とする．

図 4 に，頂点 3 の完全グラフのうち辺 2 本間に依存関係 $\{e_1, e_2\}$ がある場合の BDD を示す．

$\mathcal{E}$ 内の依存関係をすべて満たす論理関数 $F_{\mathcal{E}}$ は

$$F_{\mathcal{E}} = \bigwedge_{E' \in \mathcal{E}} F_{E'}$$

と表せる．以下では，$F_{\mathcal{E}}$ を表現する BDD の生成方法について述べる．$F_{\mathcal{E}}$ を BDD の根から順に終端の方向に向かって BDD ノードを作成する．最初は $F_{\mathcal{E}}$ の根ノードを作成し，0-枝と 1-枝を作成する．根ノードのラベルは x_{e_1} である．次に 0-枝の先のノードを作成し，ラベルを x_{e_2} とし，同様にそのノードの 0-枝と 1-枝を作成する．このように，BDD の各ノードを幅優先探索的に作成していく．i 段目のノードにはラベル x_{e_i} を付与する．

BDD のノードには，以下で述べる「依存関係の情報」を格納する．ここでは，依存関係 $E' = \{e'_1, e'_2, \ldots, e'_\ell\}$ について考える．辺の順序は $e'_1, e'_2, \ldots, e'_\ell$ であると仮定する．ラベル $x_{e'_1}$ が付けられたノードの先を作成する場合を考える．0-枝の先のノードは，$x_{e'_1} = 0$ が割り当てられることを意味する．$x_{e'_1} = 0$ という情報をそのノードに格納する．同様に，1-枝の先のノードは，$x_{e'_1} = 1$ が割り当てられることを意味するので，$x_{e'_1} = 1$ という情報をそのノードに格納する．このように，各依存関係に現れる辺について，その割当をノードに格納する．

依存関係に矛盾した割当が起こった場合，ノードの作成はせずに，0-終端に接続する．例えば，$x_{e'_1} = 0$ が格納されており，$x_{e'_2}$ のラベルが付与されているノードに対し，1-枝の先のノード ν は $x_{e'_2} = 1$ を意味し，依存関係に矛盾が起こるので，この場合は，ν を作成せずに，1-枝の先は 0-終端に接続する．最後のラベル x_{e_m} において，依存関係に矛盾が起きない場合，1-終端に接続する．

以上の手法で構築した BDD の正当性について考える．根から終端までの経路を考えるとき，依存関係に矛盾がある場合は 0-終端に到達する．依存関係に矛盾がない場合，0-終端には到達しないので，最終的に 1-終端に到達する．依存関係に矛盾がないことが，$F_{\mathcal{E}} = 1$ となるための必要十分条件であるので，構築した BDD は正しいであると言える．

3.2 実装方法

本手法では BDD の論理演算によって依存関係を考慮した信頼性 BDD を生成するが，BDD の論理演算は演算順序

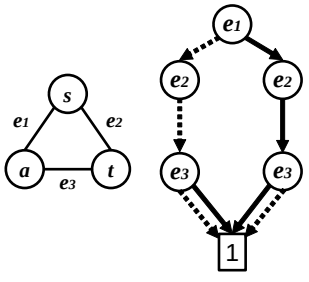

図 4 依存関係を表現する二分決定グラフの例
e_1 と e_2 に依存関係がある場合

Algorithm 1 Method1

```
1: BDD_tmp ← 依存関係を考慮しないネットワーク信頼性 BDD
2: i ← 0
3: while E が空ではない do
4:     E から E' を 1 つ取り出す
5:     BDD_i ← F_E' を表す依存関係 BDD
6:     BDD_tmp ← BDD_tmp ∧ BDD_i
7:     i ← i + 1
8: end while
9: return BDD_tmp
```

により，計算過程や出力される BDD のノード数が変化する．そのため本稿では，3 つの手法で依存関係を考慮した信頼性 BDD の実装を行う．本項ではこの 3 つの計算手法について定義する．

3.2.1 Method 1

Method 1 の手順を以下に示す．Method 1 は, 依存関係を考慮しないネットワーク信頼性 BDD BDD_{noDepend} を構築した後，$F_{E'}$ を表す BDD を生成する．この BDD_{noDepend} と依存関係 BDD の AND 演算を行い，途中計算 BDD BDD_{tmp} を生成する．ここで，2 つの BDD の AND 演算とは，BDD が表す論理関数の AND 演算を行って得られる論理関数を表す BDD を得る操作のことで，2 つの BDD ノード数の積の最悪計算量で計算できる [5]．その後，$\mathcal{E}$ から E' を取り出し，$F_{E'}$ を表す依存関係 BDD を構築し，途中計算 BDD との AND 演算を行う．これを $\mathcal{E}$ が空になるまで繰り返すことで，依存関係を全て考慮した信頼性 BDD を構築できる．

Method 1 の擬似コードを Algorithm 1 に示す．

3.2.2 Method 2

Method 2 の手順を以下に示す．Method 2 は，依存関係を考慮しないネットワーク信頼性 BDD BDD_{noDepend} を構築した後，$\mathcal{E}$ から E' と E'' を取り出し，$F_{E'}$ と $F_{E''}$ を表す 2 つの依存関係 BDD を生成する．これらの BDD の AND 演算を行い，BDD_{depend} とする．$\mathcal{E}$ が空になるまで E' を取り出し，$F_{E'}$ を表す依存関係 BDD を生成し，先の BDD_{depend} との AND 演算を取ることで，BDD_{depend} を更新していく．最後に依存関係を考慮しないネットワーク信頼性 BDD BDD_{noDepend} と，BDD_{depend} の AND 演算を行ったものを出力する．

Method 2 の擬似コードを Algorithm 2 に示す．

3.2.3 Method 3

Method 3 の手順を以下に示す．Method 3 は，依存関係を考慮しないネットワーク信頼性 BDD BDD_{noDepend} を構築した後，3.1 節で述べた手法によって $F_{\mathcal{E}}$ を表す BDD を生成する．その後，これらの BDD の AND 演算を行い，出力とする．

Method 3 の擬似コードを Algorithm 3 に示す．

Algorithm 2 Method2

1: $BDD_{\text{noDepend}} \leftarrow$ 依存関係を考慮しないネットワーク信頼性 BDD
2: $\mathcal{E}$ から E' を 1 つ取り出す
3: $BDD_{\text{depend}} \leftarrow F_{E'}$ を表す依存関係 BDD
4: $i \leftarrow 1$
5: **while** $\mathcal{E}$ が空ではない **do**
6: $\quad \mathcal{E}$ から E' を 1 つ取り出す
7: $\quad BDD_i \leftarrow F_{E'}$ を表す依存関係 BDD を生成
8: $\quad BDD_{\text{depend}} \leftarrow BDD_{\text{depend}} \wedge BDD_i$
9: $\quad i \leftarrow i + 1$
10: **end while**
11: **return** $BDD_{\text{noDepend}} \wedge BDD_{\text{depend}}$

Algorithm 3 Method3

1: $BDD_{\text{noDepend}} \leftarrow$ 依存関係を考慮しないネットワーク信頼性 BDD
2: $BDD_{\text{depend}} \leftarrow F_{\mathcal{E}}$ を表す依存関係 BDD
3: **return** $BDD_{\text{noDepend}} \wedge BDD_{\text{depend}}$

4. 依存関係を考慮した信頼性 BDD の構築実験

本章では，依存関係を考慮した信頼性ネットワーク BDD を生成する手法の計算機実験について述べる．

4.1 実装手法比較

本節では 3.2 節で述べた，依存関係を考慮したネットワーク信頼性 BDD を生成する 3 つの手法を比較する実験について述べる．

性能評価に用いるネットワークとして，ネットワークのベンチマークデータである The Internet Topology Zoo [11] の中から Switch ($n = 74, m = 92$)，UsSignal の最も大きな連結成分 ($n = 61, m = 78$) を用いる．実験で用いる依存関係集合の依存関係 1 つ辺りの長さは 2 とし，集合の要素の個数は最大 100 までとする．依存関係はランダムに決定する．試行回数は 10 回とする．なお，BDD を用いた信頼性の確率計算は BDD の構築よりも遥かに短い時間で計算が行えるため，本実験では BDD の構築のみを行い，信頼性の確率計算は行わないものとする．

計算機は Intel Xeon E5-2698 v3 (2.30GHz) の CPU と 128GB のメモリを備え，OS は Linux (Cent OS 6.8) のものを用いた．手法の実装には C++言語と TdZdd ライブラリ [12] を用いた．

まず，Switch ネットワークに対する計算結果を示す．依存関係数に対する計算時間の変化を図 5 に，BDD のノード数の変化を図 6 にそれぞれ示す．ここで，依存関係数が h のときの計算時間（または BDD のノード数）とは，依存関係の先頭から h 個を $\mathcal{E}$ としたときの Method1, Method2, Method3 の計算時間（または BDD のノード数）のことである．h を変化させたときの計算時間と BDD のノード数の変化について考察する．

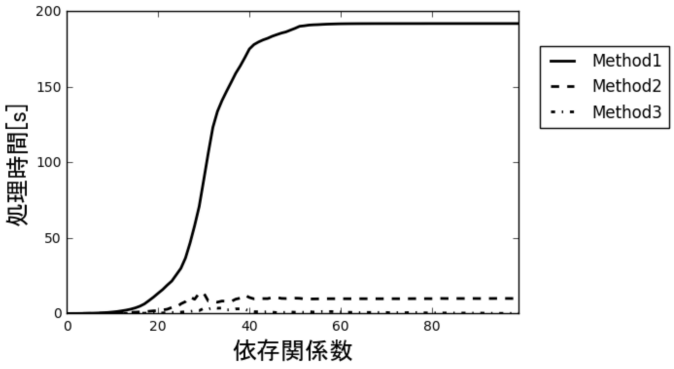

図 5 Switch 処理時間の変化

図 5 は依存関係数を変化させたときの処理時間，図 6 は依存関係数を変化させたときの BDD ノード数をそれぞれ示している．図 5 より，Method3 の処理時間が他の 2 手法と比べて速いことがわかる．これは，Method1 および Method2 が BDD の AND 演算を多く繰り返す手法である一方，Method3 は AND 演算を 1 度しか行わないからであると考えられる．一方で図 6 より，BDD のノード数は依存関係数が 30 前後で，Method3 が他手法と比べ最大になっている．ただし，依存関係数が多くなると，3 手法とも少ないノード数に収束していることがわかる．

次に，UsSignal ネットワークに対する計算結果を示す．依存関係数に対する計算時間の変化を図 7 に，BDD のノード数の変化を図 8 にそれぞれ示す．

先程と同様に，図 7 は依存関係数を変化させたときの処理時間，図 8 は依存関係数を変化させたときの BDD ノード数をそれぞれ示している．処理時間は Switch と同様に，いずれの依存関係数でも Method3 が他の 2 手法と比べて速いことがわかる．BDD のノード数も，30 前後で他 2 手法よりも大きくなるケースも存在するが，こちらも依存関係数を増やすと少ないノード数に収束している．

以上の結果から，処理時間と BDD のノード数の観点から 3 手法を評価したとき，一部の BDD ノード数を除き，基本的には Method3 が最良であるケースが多いことがわかった．次項では Method3 の手法を用いて更に大きいスケールのネットワークに対して依存関係を考慮した信頼性 BDD を生成する実験を行う．

4.2 評価実験

本項では，前項の実験で最も処理速度が速かった手法について，より大きいベンチマークネットワークに対して，依存関係を考慮した信頼性 BDD の構築を行った結果について述べる．

性能評価に用いるネットワークとして，The Internet Topology Zoo [11] の中から Deltacom ($n = 113, m = 161$)，TataNld ($n = 145, m = 186$)，UsCarrier ($n = 158, m = 189$) を用いる．依存関係の与え方は，長さ 2 の組合せをそ

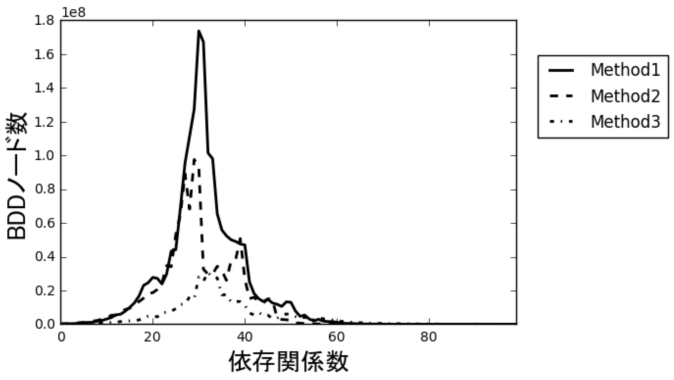

図 6 Switch BDD ノード数の変化

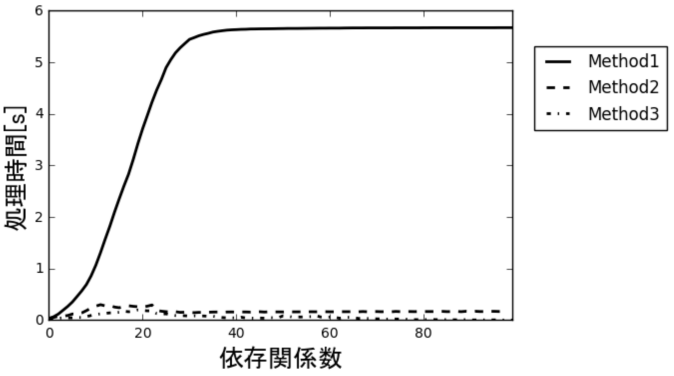

図 7 UsSignal 処理時間の変化

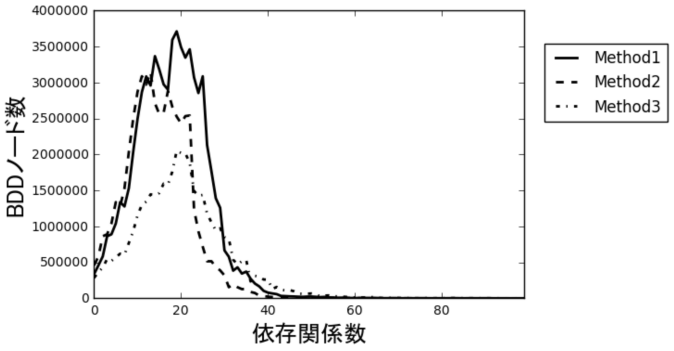

図 8 UsSignal BDD ノード数の変化

れぞれ 10, 30, 50, 100, 200, 500, 1000 個与える．これは，ある程度の傾向を見るために定めた幅である．依存関係はランダムで決定し，それぞれの組合せについて 10 回の試行を行う．なお，前項の実験と同様の理由で，信頼性の確率計算は行わないものとする．

用いた計算機は，前項の実験と同様のものを用いる．依存関係を考慮するネットワーク信頼性評価の実装は C++ 言語と TdZdd ライブラリ [12] を用いた．

計算時間を表 1 に示す．表は各ネットワークに対して，依存関係の組合せ数を変化させた際の計算時間を示している．M/O は実験環境のメモリ容量が不足し，プログラムが強制終了した回数を示している．また，Deltacom と TataNld の結果をグラフにしたものを図 9 と 10 にそれぞれ示す．

表より，いずれのネットワークに対しても依存関係が 10 程度，あるいは 500 以上であれば，依存関係を考慮した信頼性 BDD を構築できた．加えて，依存関係数が 500，1000 と多くなる程，依存関係を考慮しない信頼性 BDD と比較し少ない時間増加で計算を行えた．一方で依存関係の数が 30 から 200 程度ではいずれのネットワークに対してもメモリ容量が不足するケースが見られた．特に UsCarrier に対しては，依存関係が 30 から 100 の間では 10 回の試行回数のうち一度も BDD を構築することができなかった．

また図に示したグラフは，計算が行えたケースのみの平均値をプロットしており，メモリ容量不足により計算ができなかったケースが含まれていない．そのため厳密な数値をプロットしたグラフではないが，傾向を見るために図示している．図 9 より，Deltacom は 50 前後，TataNld は 200 前後をピークに処理時間が増している．一方で，ピークを超える依存関係数を与えたときは依存関係を考慮しないものと変わらない時間で計算を行えることが読み取れる．

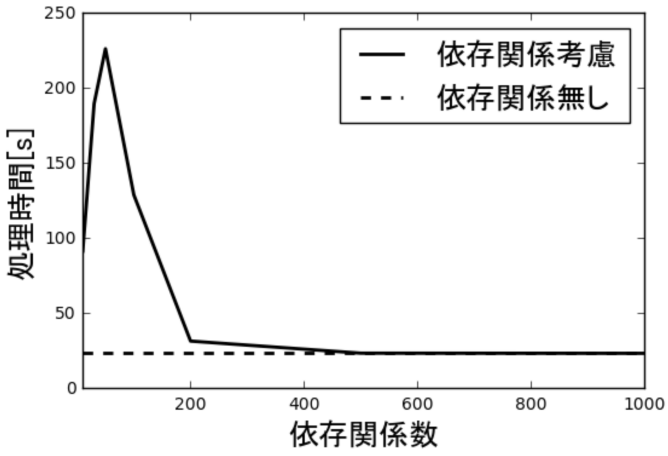

図 9 Deltacom 依存関係数と処理時間の変化

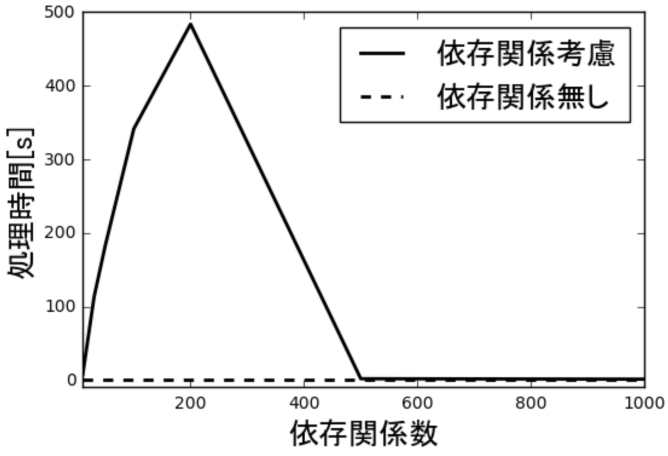

図 10 TataNld 依存関係数と処理時間の変化

表 1 依存関係を考慮した BDD の生成時間

		依存関係無し	10	30	50	100	200	500	1000
Deltacom	計算時間 [s]	23.83	91.47	189.67	226.26	128.91	31.51	23.46	23.35
	M/O	0	0	6	6	1	1	0	0
TataNld	計算時間 [s]	1.169	7.65	113.23	183.91	341.47	483.66	1.77	1.19
	M/O	0	0	2	9	9	6	0	0
UsCarrier	計算時間 [s]	113.35	403.94	-	-	-	259.48	146.92	159.92
	M/O	0	4	10	10	10	7	0	0

5. まとめ

本稿では，依存関係を考慮したネットワーク信頼性評価について述べた．依存関係を BDD で表現し，ネットワーク信頼性 BDD との論理演算を行うことで，依存関係を考慮したネットワーク信頼性 BDD を生成するという手法を提案し，3 通りの計算方法で実装した．その中で最も優れた手法を用い，$113 \leq n \leq 158$，$161 \leq m \leq 189$ 程度のネットワークに対して依存関係を考慮した信頼性 BDD の構築を行った．その結果，依存関係の数に依ってはメモリ容量不足により計算が打ち切られるケースが存在した．一方，依存関係数が増えるほど計算に必要な時間が少なくなるという結果も得られた．これは，依存関係数が多すぎる場合，1 つのリンク故障で従属する多数のリンクも同時に故障するという状況が発生するからであると考えられる．

本稿で扱った一部のネットワークでは，メモリ容量の不足により BDD の構築が失敗した．今後の研究として，依存関係を考慮したときの，より高速かつ省メモリである信頼性 BDD の構築アルゴリズムの考案が挙げられる．

参考文献

[1] 藤重 悟 編, “離散構造とアルゴリズム V,” 近代科学社, 1998.
[2] 巳波 弘佳, 井上 武, “情報ネットワークの数理と最適化,” コロナ社, 2015.
[3] L. G. Valiant, “The Complexity of Enumeration and Reliability Problems,” SIAM Journal on Computing vol. 8, no. 3, pp. 410–421, 1979.
[4] S. B. Akers, “Binary Decision Diagrams,” IEEE Transactions on Computers, vol. 100, no. 6, pp. 509–516, 1978.
[5] R. E. Bryant, “Graph-Based Algorithms for Boolean Function Manipulation,” IEEE Transactions on Computing vol. 8, no. 3, pp. 410–421, 1986.
[6] K. Sekine, H. Imai and S. Tani, “Computing the Tutte Polynomial of a Graph of Moderate Size,” in Proc. of the 6th International Symposium on Algorithms and Computation (ISAAC), pp. 224–233, 1995.
[7] G. Hardy, C. Lucet and N. Limnios, “Computing All Terminal Reliability of Stochastic Networks with Binary Decision Diagrams,” in Proc. of the 11th International Symposium on Applied Stochastic Models and Data Analysis, pp. 1468–1473, 2005.
[8] S. -Y. Kuo, F. -M. yeh and H. -Y. Lin, “Efficient and Exact Reliability Evaluation for Networks with Imperfect Vertices,” IEEE Transactions on Reliability, vol. 56, no. 2, pp. 288–300, 2007.
[9] 園田 晃己, 川原 純, 井上 武, 笠原 正治, 明石 修, 川原 亮一, 斎藤 洋, “フロンティア法によるノードの故障も考慮したネットワーク信頼性評価手法の提案,” 電子情報通信学会ネットワークシステム研究会, 信学技報, Vol. 115, No. 483, pp. 261–266, 2016.
[10] J. Kawahara, T. Inoue, H. Iwashita and S. Minato, “Frontier-Based Search for Enumerating All Constrained Subgraphs with Compressed Representation,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E100-A, no. 9, pp. 1773–1784, 2017.
[11] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden and M. Roughan, “The Internet Topology Zoo,” IEEE Journal on Selected Areas in Communications, vol. 29, no. 9, pp. 1765–1775, 2011.
[12] H. Iwashita and S. Minato, “Efficient Top-Down ZDD Construction Techniques Using Recursive Specifications,” Technical Reports, TCS-TR-A-13-69, 2013.