Vol.2022-AL-186 No.5
2022/1/28

有界モデル検査による独立集合遷移問題の解法に関する考察

戸田 貴久[1,a)]　伊藤 健洋[2]　川原 純[3]　宋 剛秀[4]　鈴木 顕[2]　照山 順一[5]

概要：独立集合遷移問題とは，グラフの独立集合に所定の変更操作を繰り返し適用して，初期の独立集合から目標の独立集合へと遷移させることができるかどうか判定する問題である．これは，もっともよく研究されている組合せ遷移問題の一つである．本研究では，変更操作の適用回数が限定された場合において，この問題に対する有界モデル検査を用いた解法を考察する．

On Solving Independent Set Reconfiguration Problems with Bounded Model Checking

Abstract: The INDEPENDENT SET RECONFIGURATION problem, which is one of the most well-studied reconfiguration problems, is to determine whether two given independent sets of a graph can be transformed into each other by repeatedly applying a prescribed operation. In this study, we consider the case where the number of applications of the operation is bounded, and propose methods to solve the case with bounded model checking.

1. はじめに

組合せ遷移は，「状態空間上での遷り変り」を数理モデル化・解析する新しいアルゴリズム理論である [1], [2]．近年，組合せ遷移への注目が集まり，計算量の解析やその背景にある数学的な性質の解明など，理論研究が活発に進められている．一方で，配電網の制御システムへの応用事例も報告されており [3]，応用研究のさらなる発展が今後期待されている．そこで本研究では，組合せ遷移において最もよく研究されている問題の一つである独立集合遷移問題を対象として，有界モデル検査を用いた実践的な解法を考察する．

独立集合遷移問題の定義は次節で与えるが，大まかには次のような問題である．入力としてグラフ G の独立集合が 2 つ与えられ，所定の変更操作を繰り返し適用することで，一方から他方へ遷移できるかどうかを判定する問題である．独立集合遷移問題は，一般に PSPACE 完全であることが知られている [4]．これは，2 つの独立集合を遷移させるために必要な変更操作の回数が，少なくとも入力サイズの超多項式回となるような問題例が存在することを暗示している．しかし，それほど多数の変更操作の適用は，現実的ではない場面も多いであろう．したがって本研究では，変更操作の適用上限回数 ℓ が与えられたとき，2 つの独立集合が ℓ 回以内の変更操作で遷移可能かどうかを判定する問題を考える．このような問題は，固定パラメータ容易性の観点から研究されており，ℓ をパラメータとしても W[1] 困難であるが，一方で XP 時間で解けることも示されている [5]．

モデル検査は，クリプキ構造として定義される状態空間において，時相論理式によって表現可能な各種の性質を検査するための手法である．モデル検査は主に設計自動化の分野において数十年に渡り研究されている．その実践的な技法の一つとして知られる**有界モデル検査**は，ハードウェアやソフトウェアのデバッギング用途など各種の問題解決のために産業界で広く活用されている．モデル検査で扱われる代表的な性質の一つは到達可能性である．それゆえ組合せ遷移とモデル検査とは親和性が高いと期待されるが，著者らの知る限り，両者の関連を明確な形で議論した文献はほとんどない．有界モデル検査に着想を得た解法として，解集合プログラミングを用いた解法 [6] が最近提案されたのみである．

本研究では，変更操作の適用上限回数 ℓ が指定された独

[1] 電気通信大学 大学院情報理工学研究科
[2] 東北大学 大学院情報科学研究科
[3] 京都大学 大学院情報学研究科
[4] 神戸大学 情報基盤センター
[5] 兵庫県立大学 大学院情報科学研究科
[a)] toda@disc.lab.uec.ac.jp

立集合遷移問題を，有界モデル検査としてモデル化する方法を与える．本研究のモデル化手法は，大きく 2 つある．1 つはシンプルな**基本モデル**であり，もう 1 つはグラフの接続関係をうまく用いる**接続行列モデル**である．接続行列モデルの手法は，**辺モデル**と**クリークモデル**の 2 つに細分される．本研究では，基本モデル，辺モデル，クリークモデルの 3 つの異なるモデルに基づく解法を比較した．実験は，DIMACS ベンチマークグラフから作成された 800 個の問題例に対して制限時間 600 秒として行ったが，クリークモデル，基本モデル，辺モデルの順に多くの問題例を解くことができた．

本稿の構成は以下の通りである．2 節では独立集合遷移問題やモデル検査の定義と用語を与える．3 節では本研究のモデル化手法を与え，4 節でその評価実験を行う．最後に 5 節でまとめる．

2. 諸定義，準備

2.1 有界な独立集合遷移問題

無向グラフ $G=(V,E)$ において，頂点部分集合 S $(\subseteq V)$ が G **の独立集合**であるとは，S のどの 2 頂点も G 上で隣接していないことをいう．今，S の各頂点には，トークン（コイン）が 1 枚ずつ置かれているとしよう．組合せ遷移では，G の独立集合間に隣接関係（変更操作）を導入し，その遷移に関する研究がなされている．特に，独立集合に対しては，次の 2 つの隣接関係を考えることが多い [4].

- **トークンジャンピング**（TJ）：G の独立集合 I と J が $|I \setminus J| = |J \setminus I| = 1$ であるとき，I と J は**（TJ の下で）隣接**しているという．すなわち，ただ 1 枚のトークンを移動することで，一方から他方が得られる．
- **トークンスライディング**（TS）：G の独立集合 I と J が $I \setminus J = \{v\}$，$J \setminus I = \{w\}$ かつ $vw \in E$ であるとき，I と J は**（TS の下で）隣接**しているという．すなわち，ただ 1 枚のトークンを G の辺 vw に沿ってスライドさせることで，一方から他方が得られる．

定義より，TJ または TS の下で隣接している 2 つの独立集合は，常に等しいサイズである．

以下では，2 つの隣接関係 TJ と TS に共通の用語を定義したいため，単に隣接関係 $\mathcal{R} \in \{\mathsf{TJ}, \mathsf{TS}\}$ と表記する．無向グラフ $G=(V,E)$ の 2 つの独立集合 I と J が**（$\mathcal{R}$ の下で）遷移可能**であるとは，次の条件 (a) と (b) を満たす G の独立集合の列 $\langle I_0, I_1, \ldots, I_\ell \rangle$ が存在することをいう．

(a) 各 $i \in \{0,1,\ldots,\ell\}$ に対し I_i は G の独立集合であり，$I_0 = I$ かつ $I_\ell = J$ である．

(b) 各 $i \in \{0,1,\ldots,\ell-1\}$ に対し，I_i と I_{i+1} は $\mathcal{R}$ の下で隣接している．

このような独立集合の列 $\langle I_0, I_1, \ldots, I_\ell \rangle$ を，I から J への（$\mathcal{R}$ 下での）**遷移列**と呼び，その**長さ**は ℓ であるとする．本稿では $\ell = 1$ の場合，すなわち I と J が隣接しているとき，I と J は 1 **ステップで遷移可能**といい，$\ell > 1$ の場合と区別することがある．

（隣接関係 $\mathcal{R}$ 下での）独立集合遷移問題とは，無向グラフ $G=(V,E)$ に対する 2 つの独立集合が与えられたとき，それらが $\mathcal{R}$ の下で遷移可能かどうか判定する問題である．また，**（隣接関係 $\mathcal{R}$ 下での）有界な独立集合遷移問題**とは，無向グラフ $G=(V,E)$ に対する 2 つの独立集合と自然数 ℓ が与えられたとき，長さが ℓ 以下の遷移列がそれらに存在するかどうか判定する問題である．

2.2 モデル検査

モデル検査は，時間とともに内部状態が非決定的に変化するシステムの動作を検査するための手法である [7], [8]. **クリプキ構造**はそのようなシステムをモデル化する仕組みであり，数学的には 4 組 (S, I, T, l) として表現される．ここで，S は状態の集合であり，I $(\subseteq S)$ は初期状態の集合であり，T $(\subseteq S \times S)$ は状態の遷移関係である．そして，$l\colon S \times \{P_0, P_1, \ldots\} \to \{0,1\}$ は各状態における原始命題の解釈を与える関数であり，各状態における命題記号の真偽を定める．

システムの動作に関して期待される性質は**仕様**と呼ばれる．仕様は時相論理を使って記述される．時相論理には大きく 2 つの種類があり，そのうち本研究で扱うのは**線形時間時相論理**（*LTL*）である．典型的な時相演算子は `X`，`F`，`G` であり，それぞれ「次の時刻」，「将来のある時刻」，「将来のすべての時刻」における論理式の真偽について言及することができる．

パスは状態の無限列であり，隣接するすべての 2 状態は遷移関係を満たすものとして定義される．パスはシステムの可能な動作の 1 つの例を表しており，パスごとに LTL の論理式（簡単に LTL 式）の真偽が定まる．

LTL **モデル検査問題**とは，LTL 式 f とクリプキ構造 M が与えられるとき，初期状態から始まるすべてのパスにおいて f が成り立つか否かを決定する問題である．LTL モデル検査は PSPACE 完全であり，一般に計算困難である．

有界モデル検査は LTL モデル検査に対する実践的な手法として広く知られている [9], [10]．基本的な考え方は，初期状態からの遷移回数を制限して，到達可能な状態集合において LTL 式が成り立たないパスを SAT ソルバーを使って探索することである．有界モデル検査の結果としてパスが返される場合は，そのパスは仕様に違反する動作例（仕様に対する**反例**）を表し，そうでない場合は，制限した範囲内に反例は存在しなかったことを意味する．

この考えに基づいて**有界モデル検査問題**は，LTL 式 f とクリプキ構造 M と自然数 n が与えられるとき，初期状態から高々 n ステップの遷移で到達可能な状態だけによっ

て構成されるパスのうち，f が成り立たないものが存在するか否かを決定する問題と書き表される．ステップ数に制限がある場合の LTL 式の意味論については [9] を参照されたい．

当初のモデル検査では検証が主目的であったが，本来的な計算困難さが実用への障壁になっていた．有界モデル検査によるデバッギングの考え方が提案され，実用規模の問題でも有界モデル検査を使うことができることが知られるようになるにつれ，有界モデル検査は産業界で広く受け入れられるようになった [*1]．

3. 提案方法

本節では，有界な独立集合遷移問題を有界モデル検査としてモデル化する 2 つの方法を与える．これらを本稿では**基本モデル**と**接続行列モデル**と呼ぶ．説明を単純にするために，以降の小節で説明される基本モデルと接続行列モデルはクリプキ構造に相当し，LTL 式は別の小節でどちらのモデルにも適用できる形で分けて説明する．

基本モデルと接続行列モデルの両方に共通するモデル化の考え方とその際に使用する記法を以下にまとめる．

2 節で解説したように，有界モデル検査の入力は LTL 式 f とクリプキ構造 $M = (S, I, T, l)$ と最大ステップ数からなる．一般に状態の個数は非常に多いので，状態として認められるものを 1 つ 1 つ明示的に指定することはしない．例えば，代表的なモデル検査器の NuSMV では，専用のモデリング言語の文法に従って変数とその型を宣言する．これにより，状態は（内部で符号化された結果として）一定の長さ n の二進列として表現されるようになる．したがって，状態の集合 S は長さ n の二進列の全体として非明示的に規定されると考えることもできるだろう．

同様に，遷移関係 T においても正しい遷移関係にある 2 つの状態の対を 1 つ 1 つ明示的に指定することはしない．その代わり，宣言された各変数について現在の時刻の値と次の時刻の値の間の関係式を記述する．本稿では以下の形式で与える．

```
next(変数) :=
 case
1）条件 1 ： 次の時刻の値を表す式;
2）条件 2 ： 次の時刻の値を表す式;
・・・
n）TRUE  ： 次の時刻の値を表す式;
    esac;
```

各条件は上の行から順に評価され，最初に成立した条件に関して，同一行（のコロンで区切られた次の項目）で指示される式によって次の時刻における変数値が定まる．最終行の TRUE は常に成り立つ．

各状態における原始命題の解釈を与える関数 l は明示的に与えない．LTL 式を構成する原始命題の例としては「変数=定数」や「変数!=定数」のような形式のものが挙げられる．状態が指定されるとき変数の値が決まるので，等号記号などの関係式の自然な解釈に従い命題の真偽が定まるからである．

3.1 基本モデル

無向グラフ $G = (V, E)$ の頂点には 1 から n まで任意の順番で数を割り当てる．ここで $n = |V|$ とする．基本モデルでは頂点とその番号を同一視することがある．独立集合のサイズを k とする．

基本モデルにおける状態 s は，それぞれ現在の時刻の独立集合，次の時刻に削除を試みる頂点，追加を試みる頂点を表す次の 3 つの要素によって構成される．

- k 個の頂点番号の列（$s.seq$ で表す）
- 1 以上 k 以下の自然数（$s.pos$ で表す）
- 1 以上 n 以下の自然数（$s.tar$ で表す）

ここで，$s.pos$ は列 $s.seq$ において次の時刻に削除する頂点が現れる位置を意味し，$s.tar$ は次の時刻に追加する頂点番号を意味する．列 $s.seq$ の i 番目の数を $s.seq[i]$ で表すことにする．

初期状態は次のように定める．初期の独立集合 I_s に対して，適当な順番でその頂点の番号を並べて得られるものを σ とする．このとき $s.seq = \sigma$ かつ $1 \leq s.pos \leq k$ かつ $1 \leq s.tar \leq n$ を満たすべての状態 s を初期状態とする．本節の冒頭で述べたように，通常のモデル検査ではこのような状態を 1 つ 1 つ初期状態として明示的に指示せず，$s.seq$ だけを定義することで初期状態の集合を定める．

遷移関係は変数ごとに次の時刻の値を表す式を指定する．ここで変数とは，$s.pos$ と $s.tar$，および各 i $(1 \leq i \leq k)$ に対して $s.seq[i]$ である．最初の 2 つの変数 $s.pos$ と $s.tar$ は各時刻において可能な値の中からランダムに決める．これにより遷移の非決定性がモデル化される．

変数 $s.seq[i]$ に対する遷移関係はトークンスライディングの場合から説明する．現在の頂点 $s.seq[i]$ は，次の時刻において $s.seq[i]$ の隣接点のうち，独立集合の他の頂点に隣接しない頂点に変更される（変更されなかった場合は $s.seq[i]$ のまま）．この遷移関係はグラフ G の構造に依存して決まるので，G に応じて条件分岐を記述する必要がある．

例として頂点集合が $V = \{1, 2, 3, 4\}$，辺集合が

$$E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}\}$$

の無向グラフ $G = (V, E)$ において，$k = 2$ の場合の変数 $s.seq[1]$ に対する遷移関係を考えよう．独立集合のもう一方の頂点を表す変数 $s.seq[2]$ に対しても同じように記述される．

[*1] 遷移回数を段階的に増やしながら有界モデル検査を繰り返すことで原理的には検証も可能になるが，現実的に検証できる問題の規模はかなり制限される．

```
next(s.seq[1]) :=
 case
1) s.pos != 1 : s.seq[1];
2) s.seq[2] = 1 & s.tar = 2     : s.seq[1];
3) s.seq[2] = 2 & s.tar = 1 | 3 : s.seq[1];
4) s.seq[2] = 3 & s.tar = 2 | 4 : s.seq[1];
5) s.seq[2] = 4 & s.tar = 3     : s.seq[1];
6) s.seq[2] = s.tar : s.seq[1];
7) s.seq[1] = 1 & s.tar = 2     : s.tar;
8) s.seq[1] = 2 & s.tar = 1 | 3 : s.tar;
9) s.seq[1] = 3 & s.tar = 2 | 4 : s.tar;
10) s.seq[1] = 4 & s.tar = 3    : s.tar;
11) TRUE : s.seq[1];
 esac;
```

1) の条件は「次の時刻に変更する頂点は $s.seq[1]$ でない」ことを意味する．よって，2) 以降の条件が評価されるときには，$s.pos = 1$ が成り立つことになる．これは $s.tar$ を次の時刻の $s.seq[1]$ の値に変更しようと試みることを意味する．

2) から 6) の条件は，独立集合のもう一方の頂点 $s.seq[2]$ が $s.tar$ と隣接するか，あるいは一致することを意味し，該当する場合がすべて列挙されている．ここで，`&`と`|`はそれぞれ論理積と論理和を意味し，例えば `s.tar = 1 | 3` は `s.tar = 1 | s.tar = 3` の略記である．よって，7) 以降の条件が評価されるときには，$s.tar$ は $s.seq[2]$ と隣接しないし，一致もしないことを意味する．このとき $s.seq[1]$ を $s.tar$ に変更することで，$s.seq$ は（現在の時刻で独立集合であるならば）次の時刻でも独立集合となることが保証される．

7) から 10) の条件は，現在変更しようと試みている頂点 $s.seq[1]$ と変更後の頂点 $s.tar$ が隣接することを意味し，該当する場合がすべて列挙されている．これらのうちどの条件が成り立つときもトークンスライディングの遷移であることが保証され，1 つの条件も成り立たないとき，最終的に 11) に到達する．

以上を一般化して，各変数 $s.seq[i]$ の遷移関係を命令文のリスト L として求める手続きが次に得られる（トークンスライディングの場合）．

(1) 空リスト L を作成する．
(2) `"s.pos != i : s.seq[i]"`を L に挿入する．
(3) 1 以上 k 以下の各 $j(\neq i)$ と各頂点 u とその閉近傍 [*2] の各点 v に対して
`"s.seq[j] = u & s.tar = v : s.seq[i]"`
を L の末尾に挿入する．
(4) 各頂点 u とその各隣接点 v に対して
`"s.seq[i] = u & s.tar = v : s.tar"`
を L の末尾に挿入する．
(5) `"TRUE : s.seq[i]"`を L の末尾に挿入する．

トークンジャンピングの場合における手続きは，4 番目のステップを取り除き，5 番目のステップにおいて挿入する命令文を`"TRUE : s.tar"`で置き換えれば得られる．

[*2] u の**閉近傍**は u あるいはその隣接点からなる．

3.2 接続行列モデル

遷移関係の記述のためには頂点の隣接関係の判定が基本になる．基本モデルでは頂点に識別子としての番号を割り当てているため，そのような番号から直接的に隣接関係を知ることができなかった．そのため，遷移関係の条件分岐において個別の変数値の可能な組合せを網羅して，正しい遷移になるときだけ $s.seq[i]$ の値を変更させるようにモデル化していた．

この問題点を解決するために，これから定義する**接続行列モデル**では，各頂点を二進列（以降，**コード**と呼ぶ）として表現し，論理演算により隣接関係を判定できるようにする．頂点コードの満たすべき性質は「2 つの頂点が隣接あるいは一致するとき，そしてそのときに限り，コードの桁ごとの論理積の結果が非ゼロとなる」である．

本稿ではこの性質を持つコードとして**接続行列**を使う．例えば，頂点と辺の接続行列では行を頂点，列を辺とみなし，u 行 e 列の要素は頂点 u に辺 e が接続するとき 1 となり，そうでないとき 0 となる．同様に，頂点とクリークの接続行列も考えることができる．すなわち，$C_1, \ldots, C_d$ を d 個のクリークとするとき，j 列をクリーク C_j に対応させて，i 行に対応する頂点が C_j に属するとき i 行 j 列の要素は 1 となり，そうでないとき 0 となる．この行列が頂点の隣接関係を正しく表現するために，隣接する任意の 2 頂点は少なくとも 1 つのクリーク C_i $(1 \leq i \leq d)$ に含まれる必要がある．この条件を満たすクリークの集合は，一般に**辺クリーク被覆**と呼ばれる．辺はサイズ 2 のクリークなので，辺集合 E も辺クリーク被覆である．

以降では，無向グラフ $G = (V, E)$ の各頂点 u のコードは，G の辺クリーク被覆に関する接続行列の u の行ベクトルによって定める．頂点コードは使用する辺クリーク被覆に依存したものとなるため，あらかじめ辺クリーク被覆を 1 つ固定した上で各頂点にコードを割り当てる．

それでは，接続行列モデルの状態，初期状態，遷移関係を順に説明していく．$n = |V|$ と表す．あらかじめ固定した辺クリーク被覆のサイズを d で表す．また，独立集合のサイズを k とする．

接続行列モデルの状態 s は，現在の時刻の独立集合 $s.seq$ と次の時刻に削除を試みる頂点（の位置）$s.pos$ と追加を試みる頂点 $s.tar$ の値によって構成される．

- 長さ d のコードを k 個並べた列（$s.seq$ で表す）
- 1 以上 k 以下の自然数（$s.pos$ で表す）
- 長さ d のコード（$s.tar$ で表す）

ここで，$s.seq$ と $s.tar$ は基本モデルとは異なるので注意が必要である．$s.seq[i]$ により i 番目のコードを表す．

初期状態は次のように定める．初期の独立集合 I_s に対して，頂点コードを任意の順番で並べることで得られるものを δ とする．このとき $s.seq = \delta$ かつ $1 \leq s.pos \leq k$ を満たし，$s.tar$ はいずれかの頂点のコードである状態 s を初期状態とする．

基本モデルの場合と同じグラフの例を使うとき，$s.seq[1]$ の遷移関係は次で与えられる．

```
next(s.seq[1]) :=
 case
1) s.pos != 1 : s.seq[1];
2) (s.seq[2] * s.tar) != v0 : s.seq[1];
3) (s.seq[1] * s.tar) != v0 : s.tar;
4) TRUE : s.seq[1];
 esac;
```

1) は基本モデルと同じなので説明を省略する．2) の条件では，$s.seq[2]$ が $s.tar$ と隣接あるいは一致することを「$s.seq[2]$ と $s.tar$ の桁ごとの論理積が非ゼロ」と表している．ここで，桁ごとの論理積を記号*，すべての桁が 0 のコードを v0 で表記している．3) の条件も同様である．

変数 $s.pos$ に対して各時刻で 1 以上 k 以下の数をランダムに決めるのは基本モデルと同じである．$s.tar$ の頂点は可能な頂点コードの中からランダムに決めるようにするが，そのようなコードは（整数の二進列表現とみたとき）一般に連続する整数とならない．そこで，すべての頂点コードを明に列挙して，その中から非決定的に選択されるように記述すれば良い．

以上を一般化して，各変数 $s.seq[i]$ の遷移関係を命令文のリスト L として求める手続きが次に得られる（トークンスライディングの場合）．

(1) 空リスト L を作成する．
(2) "s.pos != i : s.seq[i]"を L に挿入する．
(3) 1 以上 k 以下の各 j $(\neq i)$ に対して
"(s.seq[j] * s.tar) != v0 : s.seq[i]"
を L の末尾に挿入する．
(4) "(s.seq[i] * s.tar) != v0 : s.tar"
を L の末尾に挿入する．
(5) "TRUE : s.seq[i]"を L の末尾に挿入する．

この手続きは，与えられるグラフ G にほとんど依存することなく，独立集合のサイズ k と二進列の長さ d を使って各 $s.seq[i]$ の遷移関係を決めている．実質的にグラフの接続関係を参照する必要があるのは，初期状態の $s.seq$ を決定するとき，および $s.tar$ の次の時刻の値を，頂点の可能なコードの中からランダムに決めるときである．

トークンジャンピングの場合における手続きは，4 番目のステップを取り除き，5 番目のステップにおいて挿入する命令文を"TRUE : s.tar"で置き換えれば得られる．

3.3 LTL 式

基本モデルも接続行列モデルも初期状態として初期の独立集合 I_s を指定しているので，LTL 式では「将来のすべての時刻において目標の独立集合 I_t とならない」ことを記述すれば良い．なぜなら，有界モデル検査としてモデル化するとき，遷移可能な場合は LTL 式が満たされない場合に対応し，LTL 式が満たされない証拠（反例）として遷移列が出力されるからである．

はじめに，目標の独立集合 I_t になることを表す制約を考えよう．素朴な方法としては「$s.seq$ が I_t の順列のいずれかと一致する」ことを制約として記述すれば良いが，$k!$ 通りの場合分けが生じるため現実的ではない．実は，各 $s.seq[i]$ に対して「$s.seq[i]$ が目標の独立集合 I_t のいずれかの頂点と一致する」という制約だけで実質的に同じ効果のある制約となる．なぜなら，基本モデルも接続行列モデルも遷移の途中で重複する頂点は生じないからである．

例えば，目標の独立集合 I_t は 2 つの頂点からなり，$v2, v3$ はそれぞれの頂点の表現とする（基本モデルの場合は頂点番号，接続行列モデルの場合は頂点コード）．このとき上で述べた制約は次で表される．

```
   (s.seq[1] = v2 | s.seq[1] = v3)
 & (s.seq[2] = v2 | s.seq[2] = v3)
```

このように表される制約 P の否定!P に対して時相演算子 G を適用して得られる LTL 式 G!(P) は「将来のすべての時刻において I_t とならない」ことを表す．

4. 実験

前節で導入したモデルに基づく有界な独立集合遷移問題に対する 3 つの手法を計算機実験により比較して，性能を評価する．

4.1 実験の設定

提案手法では有界な独立集合遷移問題を有界モデル検査問題としてモデル化して，有界モデル検査器を適用することで元の独立集合遷移問題を解いている．手法ごとの違いは有界モデル検査問題へのモデル化だけであり，本実験では以下の 3 つのモデルを比較することにする．

- 基本モデル
- 辺に関する接続行列モデル（簡単のため，**辺モデル**と呼ぶ）
- 辺クリーク被覆に関する接続行列モデル（簡単のため，**クリークモデル**と呼ぶ）

ここで，接続行列モデルに関して，頂点コードとして頂点と辺の接続行列を使う場合と，頂点と（あらかじめ求めた辺クリーク被覆に属する）クリークの接続行列を使う場合

表 1　ベンチマークグラフ.
Table 1　Benchmark Graphs.

	頂点数	辺数	辺クリーク被覆サイズ
MANN_a27	378	702	468
MANN_a9	45	72	48
c-fat200-1	200	18,366	323
hamming6-2	64	192	192
johnson16-2-42	120	1,680	16
johnson32-2-4	496	14,880	32
johnson8-4-4	70	560	78
keller4	171	5,100	464

の 2 通りを考える．以降では，提案手法をこれらのモデルの名称で言及することがある．

提案手法の中で使用した外部プログラムは次の通りである．有界モデル検査器として NuSMV 2.6.0 [*3] [11] を使用した．NuSMV の内部で利用する SAT ソルバーとして MiniSat 2.2.0 [*4] [12] を指定した．辺クリーク被覆を計算するプログラムとして ECC [*5] [13] を使用した．

本実験で使用する問題例は次のように作成した（TJ モデルに対して 800 個，TS モデルに対して 800 個）．**使用グラフ**： 第 2 回 DIMACS Implementation Challenge [*6] において使用されたベンチマークグラフに対して，ECC によって計算された辺クリーク被覆のサイズが 500 未満のものを使用した．ただし，hamming6-4 と johnson8-2-4 も辺クリーク被覆のサイズが 500 未満であったが，初期の独立集合と目標の独立集合の対を（後で述べる方法で）生成できなかったので，この二つのグラフは除外されている．各グラフの情報を表 1 に示す．

独立集合の対：各グラフに対して初期の独立集合と目標の独立集合の対を次の方法により，相異なる 100 個の対を作成した．

(1) グラフから頂点を 1 つずつランダムに選び，サイズ 5 の独立集合を作成する（独立でなければ，最初から選び直す）；これを初期の独立集合とする．
(2) 独立集合に対してランダムに遷移をさせて，独立集合を更新することを繰り返す．
(3) この結果として得られる独立集合を目標の独立集合とする．

ここで，トークンジャンピング（TJ）に基づく独立集合遷移問題の作成では，TJ の遷移に従うようにした．同様に，トークンスライディング（TS）の場合は TS の遷移に従うようにした．

遷移列の長さの最大値：有界な独立集合遷移問題において入力として指定する遷移列の長さの最大値は今回作成したすべての問題例で 10 とした．

[*3] https://nusmv.fbk.eu/
[*4] http://minisat.se/
[*5] https://github.com/Pronte/ECC
[*6] http://www.dimacs.rutgers.edu/programs/challenge/

制限時間は 600 秒とした．制限時間内に，遷移列を発見できるか，あるいは指定した最大長さ以下の遷移列が存在しないことを報告する場合に，問題例を解くことができたことにする．

実験の環境は以下の通りである．OS: Red Hat Enterprise Linux 8.3，CPU: Intel® Xeon® Gold 5218 (2.30GHz)，主メモリ: 2TB.

4.2　実験結果と考察

図 1 は TJ モデルの問題例（800 個）に対して 3 つの手法の計算時間を比較した結果である．図 2 は TS モデルの問題例（800 個）に対して同様に比較した結果である．これらの図では，各手法に対して，解くことができた問題例が，計算時間の昇順になるようにプロットされている．このような図は**カクタスプロット**と呼ばれており，SAT ソルバーの性能評価などにおいて標準的に使われている．

例えば，y 座標を特定の値（例えば 200）で固定して，水平線 $y = 200$ と各手法の点の系列（を補完して得られる曲線）との交点の x 座標を確認することで，200 秒以内に解くことができた問題例の個数を確認することができる．

図 1 と 2 のどちらも，概ねどの y 座標の値に対しても，辺モデルより基本モデルやクリークモデルの方が圧倒的に多くの問題例を解くことができている．また，基本モデルとクリークモデルの差も顕著に現れている．

同じ接続行列モデルでも辺モデルとクリークモデルでこれ程大きな性能差が生じた理由は，辺の個数による影響があったと考えられる．実際，両モデルの間の実質的な違いは頂点コードの長さであり，頂点コードの長さは辺モデルでは辺の個数，クリークモデルでは辺クリーク被覆のサイズとなるが，表 1 に示されているようにグラフの辺数に比べて，辺クリーク被覆のサイズがかなり小さくなっている．基本モデルは辺の個数に直接影響を受けないモデル化であるので，辺モデルより効率的であったと考えられる．

以上のことから，一定の制限時間を設けたときには，クリークモデルがもっとも多くの問題例を解くことができ，その後に基本モデル，辺モデルの順に続くことが分かる．特に本実験でプログラムの最大動作時間として設定した 600 秒以内に解くことができた問題例の数は各手法の系列の最後の点の x 座標であり，クリークモデルが他のモデルに比べて極めて多くの問題例を解いていることが分かる．

5.　おわりに

本研究では，有界な独立集合遷移問題に対して，有界モデル検査を用いた実践的な解法を考察した．この問題を有界モデル検査問題としてモデル化する 3 つの手法を提案し，既存の有界モデル検査器を適用できるようにした．計算機実験の結果，クリークモデルを用いると，他 2 つのモデルに比べて，多くの問題例が解けることが分かった．こ

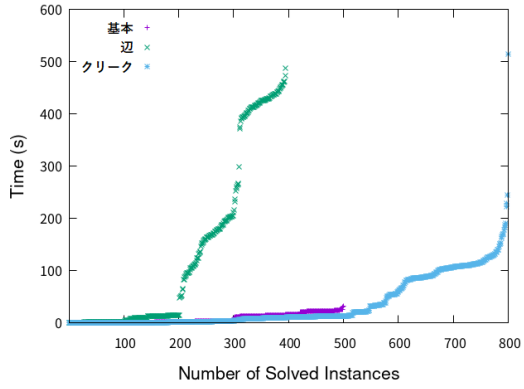

図 1 計算時間の比較（TJ モデル）.
Fig. 1 Comparison of computation time (TJ-model).

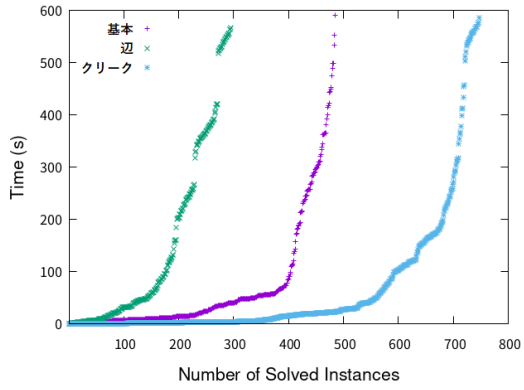

図 2 計算時間の比較（TS モデル）.
Fig. 2 Comparison of computation time (TS-model).

の理由としては，今回使用した DIMACS ベンチマークグラフの中には，辺の個数に比べて辺クリーク被覆のサイズがかなり小さいものがあったため，クリークモデルの接続行列のサイズが，辺モデルの接続行列のサイズに比べて十分小さくなったことが考えられる．他方，グラフがそれほど大きくない場合は接続行列モデルよりも基本モデルの方が効率的な場合が多い．接続行列モデルは各頂点にコードとして多くの情報を持つため，小さいグラフではこれがオーバーヘッドになると考えられる．

謝辞

本研究は JSPS 科研費 JP17K17725, JP18H04091, JP18K18003, JP19K11814, JP20H05793, JP20H05794, JP20K11666, JP20K11748 の助成を受けたものである．

参考文献

[1] Ito, T., Demaine, E. D., Harvey, N. J., Papadimitriou, C. H., Sideri, M., Uehara, R. and Uno, Y.: On the complexity of reconfiguration problems, *Theoretical Computer Science*, Vol. 412, No. 12, pp. 1054–1065 (online), DOI: 10.1016/j.tcs.2010.12.005 (2011).
[2] Nishimura, N.: Introduction to Reconfiguration, *Algorithms*, Vol. 11, No. 4 (online), DOI: 10.3390/a11040052 (2018).
[3] 東北大学，明電舎共同プレスリリース：配電損失の最適化と切替手順の同時算出が可能に—電力の発送電分離を目前に自動計算の手法開発，2019 年 6 月 27 日配信.
[4] Kamiński, M., Medvedev, P. and Milanič, M.: Complexity of independent set reconfigurability problems, *Theoretical Computer Science*, Vol. 439, pp. 9–15 (online), DOI: 10.1016/j.tcs.2012.03.004 (2012).
[5] Mouawad, A. E., Nishimura, N., Raman, V., Simjour, N. and Suzuki, A.: On the Parameterized Complexity of Reconfiguration Problems, *Algorithmica*, Vol. 78, No. 1, pp. 274–297 (online), DOI: 10.1007/s00453-016-0159-2 (2017).
[6] 山田悠也，　湊真一，番原睦則：解集合プログラミングに基づく組合せ遷移ソルバーの実装方式に関する考察，日本ソフトウェア科学会第 38 回大会講演論文集 (2021).
[7] Clarke, E. M., Emerson, E. A. and Sifakis, J.: Model Checking: Algorithmic Verification and Debugging, *Commun. ACM*, Vol. 52, No. 11, pp. 74–84 (online), DOI: 10.1145/1592761.1592781 (2009).
[8] Clarke, E. M., Grumberg, O. and Peled, D. A.: *Model Checking*, MIT Press, Cambridge, MA, USA (2000).
[9] Biere, A., Cimatti, A., Clarke, E. and Zhu, Y.: Symbolic Model Checking without BDDs, *Tools and Algorithms for the Construction and Analysis of Systems* (Cleaveland, W. R., ed.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 193–207 (online), DOI: 10.1007/3-540-49059-0_14 (1999).
[10] Biere, A., Biere, A., Heule, M., van Maaren, H. and Walsh, T.: *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*, IOS Press, NLD (2009).
[11] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. and Tacchella, A.: NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking, *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, LNCS, Vol. 2404, Copenhagen, Denmark, Springer, pp. 359–364 (online), DOI: 10.1007/3-540-45657-0_29 (2002).
[12] Eén, N. and Sörensson, N.: An Extensible SAT-solver, *Theory and Applications of Satisfiability Testing* (Giunchiglia, E. and Tacchella, A., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 502–518 (online), DOI: 10.1007/978-3-540-24605-3_37 (2004).
[13] Conte, A., Grossi, R. and Marino, A.: Large-scale clique cover of real-world networks, *Information and Computation*, Vol. 270, p. 104464 (online), DOI: 10.1016/j.ic.2019.104464 (2020).