解集合プログラミングを用いた配電網問題の解法に関する一考察

山田 健太郎　湊 真一　番原 睦則

配電網問題は求解困難な組合せ最適化問題の一種である．配電網問題は，トポロジ制約と電気制約を満たしつつ，電力の損失を最小にするスイッチの開閉状態を求めることが目的である．トポロジ制約のみの配電網問題は，与えられた連結グラフと根と呼ばれるノードから根付き全域森を探索する部分グラフ探索問題 (根付き全域森問題) に帰着できる．本稿では，解集合プログラミング (ASP) を用いた根付き全域森問題の解法について述べる．考案した ASP 符号化は，根付き全域森問題の連結制約を ASP の個数制約で表現することにより，基礎化後のルール数を少なく抑えるよう工夫されており，大規模な問題に対する有効性が期待できる．また，配電網における障害時の復旧予測への応用を狙いとし，根付き全域森問題の 2 つの実行可能解が，局所的な変形による遷移だけで互いに移りあえるかを問う “解の遷移問題” への拡張についても述べる．最後に，実用規模の問題を含む問題集を用いた実験結果について報告する．

1　はじめに

電力網の構成制御は，エネルギーの節約や安定した電力供給を支える重要な研究課題である．電力網は，高電圧で発電所と変電所を結ぶ**送電網**と，低電圧で変電所と家庭や工場といった需要家を結ぶ**配電網**に分類される．配電網は変電所と需要家との間で構成される電力供給ネットワークであり，その構成技術はスマートグリッドや，災害時の障害箇所の迂回構成などを支える重要な基盤技術である．

配電網問題は，供給経路に関する**トポロジ制約**と，電流・電圧に関する**電気制約**を満たしつつ，電力の損失を最小にするスイッチの開閉状態を求めることが目的である [4]．トポロジ制約は，短絡 (供給経路上のループ，複数の変電所と結ばれる需要家) と停電 (変電所と結ばれない需要家) が発生しないことを保証する．電気制約は，供給経路の各区間で許容電流を超えないこと，電気抵抗による電圧降下が許容範囲を超えないことを保証する．本稿ではトポロジ制約のみの配電網問題を対象とする．

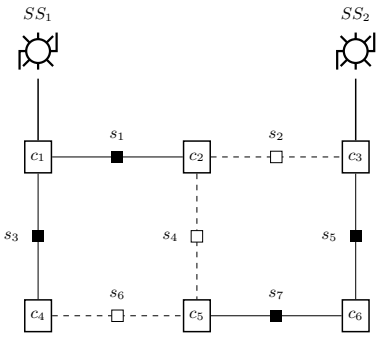

図 1　配電網問題の例

トポロジ制約のみの配電網問題の例を図 1 に示す．この例は，2 つの変電所 $\{SS_1, SS_2\}$，7 個のスイッチ $\{s_1, \ldots, s_7\}$，6 つの需要家 $\{c_1, \ldots, c_6\}$ から構成されている．■ は閉じたスイッチ，□ は開いたスイッチを表している．配電網問題の実行可能解は閉じたスイッチの集合で表すことができる．この例は実行可能解 $\{s_1, s_3, s_5, s_7\}$ を表している．需要家 $\{c_1, c_2, c_4\}$

A Study on Solving Power Distribution Network Problem with Answer Set Programming.
Kentaro Yamada, Mutsunori Banbara, 名古屋大学大学院情報学研究科, Graduate School of Informatics, Nagoya University.
Shin-ichi Minato, 京都大学大学院情報学研究科, Graduate School of Informatics, Kyoto University.

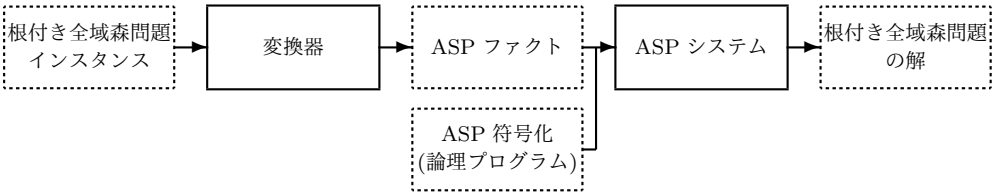

図 2　提案アプローチの構成図

は変電所 SS_1 から，需要家 $\{c_3, c_5, c_6\}$ は変電所 SS_2 から電力を供給され，トポロジ制約を満たしていることがわかる．

配電網問題は求解困難な組合せ最適化問題の一種であり，これまでフロンティア法を用いた解法等が提案されている [6]．トポロジ制約のみの配電網問題は，与えられた連結グラフと根と呼ばれる特殊なノードから，**根付き全域森**を求める部分グラフ探索問題に帰着できることが知られている [7]．以降，この問題を**根付き全域森問題**と呼ぶ．

解集合プログラミング (Answer Set Programing; ASP [1] [3] [5] [8]) は，論理プログラミングから派生したプログラミングパラダイムである．ASP 言語は，一階論理に基づく知識表現言語の一種であり，論理プログラムは ASP のルールの有限集合である．ASP システムは論理プログラムから安定モデル意味論 [3] に基づく解集合を計算するシステムである．近年，SAT ソルバーの技術を応用した高速な ASP システムが確立され，制約充足問題，プランニング，システム生物学，時間割問題，システム検証など様々な分野への実用的応用が急速に拡大している [2]．

本稿では，解集合プログラミングを用いた根付き全域森問題の解法について述べる．提案アプローチでは，まず与えられた問題インスタンスを ASP のファクト形式に変換した後，ASP ファクトと根付き全域森問題を解くための ASP 符号化と結合した上で，高速 ASP システムを用いて解を求める (図 2 参照)．

根付き全域森問題を解く ASP 符号化 (論理プログラム) として，基本符号化と改良符号化の 2 種類を考案した．特に，改良符号化は，根付き全域森問題の連結制約を ASP の個数制約で表現することにより，基礎化後のルール数を少なく抑えるよう工夫されており，大規模な問題に対する有効性が期待できる．

また，配電網における障害時の復旧予測への応用を狙いとし，根付き全域森問題の 2 つの実行可能解が，局所的な変形による遷移だけで互いに移りあえるかを問う “解の遷移問題” への拡張を行なった．この遷移問題を解くには，複数の根付き全域森問題を繰り返し解く必要がある．しかし，各問題中の制約の大部分は共通であるため，ASP システムが同一の探索空間を何度も調べることになり，求解効率が低下するという問題点がある．この問題を解決するために，マルチショット ASP 解法を用いた実装を提案する．この解法は，ASP システムが同様の探索失敗を避けるために獲得した学習節を (部分的に) 保持することで，無駄な探索を行うことなく，制約を追加した論理プログラムを連続的に解くことができる．そのため，求解性能の向上が期待できる．

提案アプローチの有効性を評価するために，DNET (Power Distribution Network Evaluation Tool) †1 に公開されている問題集と，Graph Coloring and its Generalizations †2 に公開されているグラフ彩色問題を基に生成した根付き全域森問題を用いて，実行実験を行なった．その結果，改良符号化は，基本符号化と比較して，より多くの問題をより高速に解くことができた．また，改良符号化は辺数 (スイッチ数) が 40,000 個を超える問題も解いており，大規模問題に対する ASP の有効性が確認できた．

解の遷移問題については，DNET で公開されている実用規模の問題 (fukui-tepco) をベースとした問題集を用いて実験を行った．その結果，マルチショット ASP 解法は，通常の解法と比較して，平均で 3.3 倍の高速化を実現し，マルチショット ASP 解法の優位

†1 `https://github.com/takemaru/dnet`
†2 `https://mat.tepper.cmu.edu/COLOR04/`

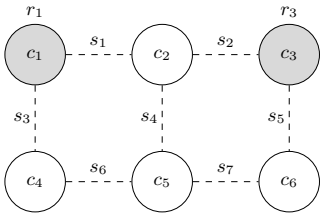

図 3　根付き全域森問題の入力例

性が確認できた．

本稿の構成は以下の通りである．2 節で根付き全域森の定義を示し，3 節で解集合プログラミングの説明を行う．4 節で根付き全域森問題の ASP 符号化を示し，5 節で根付き全域森問題を遷移問題へ拡張を行う．6 節で評価実験とその考察を述べ，最後に，7 節で本稿をまとめる．

2　根付き全域森問題

根付き全域森は以下のように定義される [7]．

定義． グラフ $G = (V, E)$ と，根と呼ばれる V 上のノードの集合が与えられたとする．このとき，G 上の根付き全域森とは，以下の制約を満たす G の部分グラフ $G' = (V, E'), E' \subseteq E$ である．

1. G' はサイクルを持たない．(非閉路制約)
2. G' の各連結成分は，ちょうど 1 つの根を含む．(根付き連結制約)

本稿では，与えられたグラフ G から，根付き全域森 G' を求める部分グラフ探索問題を**根付き全域森問題**と呼ぶ．

根付き全域森問題の入力例となるグラフを図 3 に示す．図 3 は，図 1 で示した配電網に対応しており，需要家 $\{c_1, \ldots, c_6\}$ は，**ノード**に対応し，スイッチ $\{s_1, \ldots, s_7\}$ は，**辺**に対応する．また，図中の色付きノード $\{r_1, r_3\}$ は変電所と直接繋がっている需要家を意味しており，**根**に対応している．

根付き全域森の例を図 4 に示す．根付き全域森は，各連結成分が必ずちょうど 1 つの根をもつ木構造を形成することで，非閉路制約と根付き連結制約を満たす．図 4 の上側は，図 1 の配電網問題の解に対応している．また，根付き全域森問題には解が複数存在し得る．例えば，図 4 の下側は，ある連結成分が根のみとなる場合を表している．

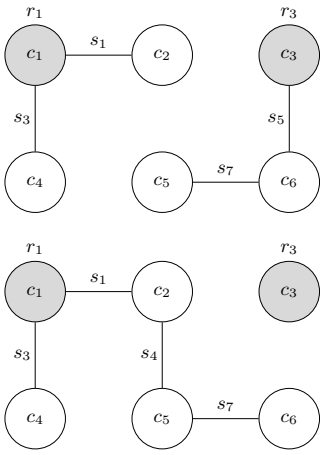

図 4　根付き全域森の出力例 (2 つ)

3　解集合プログラミング

ASP の言語は，一般拡張選言プログラムをベースとしている [5]．本稿では，説明の簡略化のため，そのサブクラスである標準論理プログラムについて説明する．以降，標準論理プログラムを単に論理プログラムと呼ぶ．

論理プログラムは，以下の形式の**ルール**の有限集合である．

$$a_0 \texttt{ :- } a_1\texttt{ , }\ldots\texttt{ , }a_m\texttt{ , not } a_{m+1}\texttt{ , }\ldots\texttt{ ,not } a_n\texttt{ .}$$

ここで，$0 \leq m \leq n$ であり，各 a_i はアトム，`not` は**デフォルトの否定** †3，“,” は連言を表す．“:-” の左側を**ヘッド**，右側を**ボディ**と呼ぶ．“.” はルールの終わりを表す終端記号である．ルールの直観的な意味は，「$a_1, \ldots, a_m$ がすべて成り立ち，$a_{m+1}, \ldots, a_n$ のそれぞれが成り立たないならば，a_0 が成り立つ」である．ボディが空のルール (すなわち a_0 :- .) を**ファクト**と呼び，“:-” を省略することができる．

ヘッドが空のルールを**一貫性制約**と呼ぶ．

$$\texttt{:- } a_1\texttt{ , }\ldots\texttt{ , }a_m\texttt{ , not } a_{m+1}\texttt{ , }\ldots\texttt{ ,not } a_n\texttt{ .}$$

例えば，一貫性制約 “:- a_1,a_2.” は,「a_1 と a_2 が両方同時に成り立つことはない」を意味し，“:- a_1, not a_2.” は,「a_1 が成り立つならば，a_2 も成り立つ」を意味

†3　**失敗による否定**とも呼ばれる．述語論理で定義される否定 (¬) とは意味が異なる．

```
node(1). node(2). node(3). node(4). node(5). node(6).

root(1).  root(3).

edge(1,2).  edge(1,4). edge(2,5).  edge(2,3).
edge(3,6).  edge(4,5). edge(5,6).
```

コード 1　根付き全域森問題 (図 3) のファクト表現

する．

ASP 言語には，組合せ問題を解くために便利な拡張構文が用意されている．その代表的なものが**選択子**と**個数制約**である．例えば，選択子 “{a_1;...;a_n}.” をファクトとして書くと，「アトム集合 $\{a_1,\ldots,a_n\}$ の任意の部分集合が成り立つ」を意味する．個数制約は選択子の両端に選択可能な個数の上下限を付けたものである．例えば，“:- a_0, not lb {a_1;...;a_n} ub.” と書くと，「a_0 が成り立つならば，$a_1,\ldots,a_n$ のうち，lb 個以上 ub 個以下が成り立つ」を意味する．また，**重み付き個数制約** (#sum) も用意されている．例えば，“:- a_0, not lb #sum {w_1:a_1;...;w_n:a_n} ub.” と書くと，「a_0 が成り立つならば，$a_1,\ldots,a_n$ のうち真となるアトムの重み和が lb 個以上 ub 個以下である」を意味する．項 w_i は重みを表す．

ASP システムは，与えられた論理プログラムから，安定モデル意味論 [3] に基づく解集合を計算するシステムである．本稿では，高性能かつ高機能な ASP システムとして世界中で広く使われている *clingo* †4 を使用する．*clingo* を含め最新の高速 ASP システムは，グラウンダーを用いて変数を含む論理プログラムを変数を含まない論理プログラムに変換 (**基礎化**) したのち，ASP ソルバーを用いて解集合を計算する方式が主流となっている．

4　根付き全域森問題の ASP 符号化

本節では，根付き全域森問題の入力と制約を論理プログラムとして表現する方法について述べる．

ASP ファクト形式. 根付き全域森問題の入力 (図 3) のファクト表現をコード 1 に示す．この問題は，ノード 6 個，根ノード 2 個，辺 7 個から構成され，それぞれ，アトム node/1，root/1，edge/2 によって表されている．例えば，ファクト edge(1,2). は，ノード 1 とノード 2 が隣接していることを表す．

```
1  %% solution candidates
2  { inForest(X,Y) } :- edge(X,Y).
3
4  %% reachability
5  reached(R,R) :- root(R).
6  reached(X,R) :- reached(Y,R), inForest(Y,X).
7  reached(X,R) :- reached(Y,R), inForest(X,Y).
8
9  %% acyclicity constraint
10 :- root(R),
11    not 1 #sum{ 1,X:reached(X,R) ;
12                -1,X,Y:inForest(X,Y),reached(X,R),reached(Y,R)
13              } 1.
14
15 %% rooted connectivity constraint
16 :- node(X),  not reached(X,_).
17 :- reached(X,R1), reached(X,R2), R1 < R2.
```

コード 2　根付き全域森問題の基本符号化

```
1  %% solution candidates
2  { inForest(X,Y) } :- edge(X,Y).
3
4  %% reachability
5  reached(R,R) :- root(R).
6  reached(X,R) :- reached(Y,R), inForest(Y,X).
7  reached(X,R) :- reached(Y,R), inForest(X,Y).
8
9  %% acyclicity constraint
10 :- root(R),
11    not 1 #sum{ 1,X:reached(X,R) ;
12                -1,X,Y:inForest(X,Y),reached(X,R),reached(Y,R)
13              } 1.
14
15 %% rooted connectivity constraint
16 :- node(X), not 1 { reached(X,R) } 1.
```

コード 3　根付き全域森問題の改良符号化

基本符号化. 根付き全域森問題の ASP 符号化をコード 2 に示す．2 行目のルールは，各辺 (X,Y) に対して，解の候補となるアトム inForest(X,Y) を導入する．この inForest(X,Y) は，辺 (X,Y) が根付き全域森

†4　https://potassco.org/

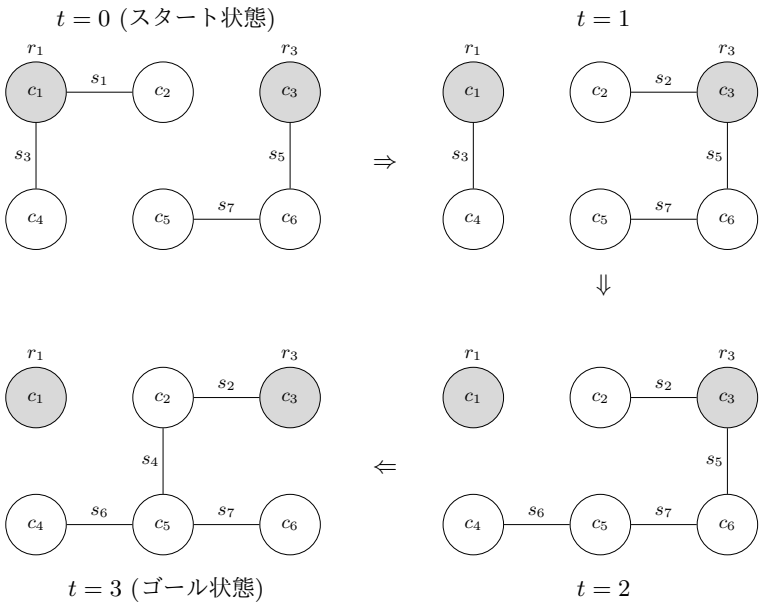

図 5　根付き全域森遷移問題 (遷移制約 $d = 2$) の解の一例

に含まれることを意味する．各ノードの到達可能性は5〜7 行目のルールで表される．アトム `reached(X,R)` は，ノード `X` は根 `R` から到達可能であることを意味する．5 行目のルールは，各根ノードは自分自身から到達可能であることを表す．6〜7 行目のルールは，ノード `Y` が根ノード `R` から到達可能であり，かつ，辺 (`X`,`Y`) が全域森に含まれるならば，ノード `X` も `R` から到達可能であることを表す．

非閉路制約は 10〜13 行目のルールで表される．このルールは，各連結成分のノード数と辺数の差が 1 になること (木の性質) を，ASP の重み付き個数制約を使って表している．根付き連結制約は 16〜17 行目のルールで表される．16 行目のルールは，各ノードは少なくとも 1 つの根から到達可能であることを表している (*at-least-one* 制約)．17 行目のルールは，各ノードは高々1 つの根から到達可能であることを表している (*at-most-one* 制約)．これら 2 つの一貫性制約により，各ノードはちょうど 1 つの根から到達可能であることが強制される．

改良符号化. 基本符号化は，根付き全域森問題の制約を ASP のルール 7 個で簡潔に表現できる．しかし，根付き連結制約を表す *at-most-one* 制約の基礎化後のルール数は，根ノード数の 2 乗に比例するため，大規模な問題に対する求解性能が低下する可能性がある．この問題を解決するために考案した改良符号化をコード 3 に示す．基本符号化との違いは，根付き連結制約を ASP の個数制約で表している点である (16 行目)．グラフのノード数を n，根ノード数を r として，根付き連結制約の基礎化後のルール数を比較すると，基本符号化が $n(1 + {}_rC_2)$ 個なのに対し，改良符号化は n 個と少なく抑えることができる．これにより，大規模な問題に対する有効性が期待できる．

5　遷移問題への拡張

配電網の構成制御における障害時の復旧予測への応用を狙いとし，ある初期配電網構成 (スタート状態) から目的配電網構成 (ゴール状態) へのスイッチの切替手順を求める遷移問題を考える．各ステップ t で切替可能なスイッチの個数を d 個以下に制限し，最短ステップ長での切替手順を求めることが目的である．

トポロジ制約のみを考える場合，この配電網の遷移問題は，根付き全域森問題とその 2 つの実行可能解が与えられたとき，ある解から他のもう一つの解へ根付き全域森の制約を満たしながら移る “解の遷移問題” に帰着できる．各ステップ t で変更可能な辺の数を d 個以下に制限し (**遷移制約**)，最短ステップ長で

```
%% timestep
t(0..t).

%% start
:- not inForest(X,Y,0), init_Forest(X,Y).

%% goal
:- not inForest(X,Y,t), goal_Forest(X,Y).

%% solution candidates
{ inForest(X,Y,T) } :- edge(X,Y), t(T).

%% reachability
reached(R,R,T) :- root(R), t(T).
reached(X,R,T) :- reached(Y,R,T), inForest(Y,X,T), t(T).
reached(X,R,T) :- reached(Y,R,T), inForest(X,Y,T), t(T).

%% acyclicity constraint
:- root(R), t(T),
   not 1 #sum {  1,X:   reached(X,R,T);
                -1,X,Y: inForest(X,Y,T),
                        reached(X,R,T),
                        reached(Y,R,T)
               } 1.

%% rooted connectivity constraint
:- node(X), t(T), not 1 { reached(X,R,T) } 1.

%% transition constraint
dist(X,Y,T) :-
    inForest(X,Y,T), not inForest(X,Y,T-1), t(T), T>0.
dist(X,Y,T) :-
    inForest(X,Y,T-1), not inForest(X,Y,T), t(T), T>0.
:- t(T), not #sum{ 1,X,Y:dist(X,Y,T) } d.
```

コード 4　全域森遷移問題のシングルショット符号化

の変更手順を求めることが目的となる．この遷移問題を**根付き全域森遷移問題**と呼ぶことにする．

図 3 の根付き全域森問題とその 2 つの実行可能解から構成された根付き全域森遷移問題の解の一例を図 5 に示す．各ステップ t で変更可能な辺の数を $d = 2$ 以下に制限している．この解のステップ長は 3 であり，スタート状態 ($t = 0$) からゴール状態 ($t = 3$) まで，根付き全域森の制約を満たしながら遷移していることがわかる．

シングルショット符号化. 根付き全域森遷移問題の ASP 符号化をコード 4 に示す．この符号化は，与えられた根付き全域森遷移問題に対して，ステップ長 `t` の解が存在するかを判定し，存在する場合はその解を返す論理プログラムである．根付き全域森問題の改良符号化 (コード 3) からの拡張点は以下の通りである．

- 新しくステップを表すアトム `t(T)` を導入，
- 改良符号化の各アトムにステップを表す項 `T` を引数として追加，

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#program base.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% start
:- not inForest(X,Y,0), init_Forest(X,Y).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#program step(t).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% timestep
t(t).

%% solution candidates
{ inForest(X,Y,t) } :- edge(X,Y), t(t).

%% reachability
reached(R,R,t) :- root(R), t(t).
reached(X,R,t) :- reached(Y,R,t), inForest(Y,X,t), t(t).
reached(X,R,t) :- reached(Y,R,t), inForest(X,Y,t), t(t).

%% acyclicity constraint
:- root(R), t(t),
   not 1 #sum {  1,X:   reached(X,R,t);
                -1,X,Y: inForest(X,Y,t),
                        reached(X,R,t),
                        reached(Y,R,t)
               } 1.

%% rooted connectivity constraint
:- node(X), t(t), not 1 { reached(X,R,t) } 1.

%% transition constraint
dist(X,Y,t) :-
    inForest(X,Y,t), not inForest(X,Y,t-1), t(t), t>0.
dist(X,Y,t):-
    inForest(X,Y,t-1), not inForest(X,Y,t), t(t), t>0.
:- t(t), not #sum{ 1,X,Y:dist(X,Y,t) } d, t>0.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#program check(t).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% goal
:- not inForest(X,Y,t), goal_Forest(X,Y), query(t).
```

コード 5　全域森遷移問題のマルチショット符号化

- スタート状態とステップ 0，ゴール状態とステップ t を対応させるルールを追加，
- 遷移制約を表すルールを追加．

1 行目の t(0..t). は，t(0)., t(1).,... t(t). に展開され，各ステップの識別子を表す．定数 t はステップ長を表す整数値であり，実行時に与えられる．

スタート状態はファクト init_Forest/2 の集合で与えらる．5 行目のルールは，スタート状態とステップ 0 での根付き全域森が一致することを強制する．同様にして，8 行目のルールは，ゴール状態とステップ t での根付き全域森が一致することを強制する．

遷移制約は 30〜34 行目のルールで表される．dist(X,Y,T) はステップ T-1 とステップ T の間で辺 (X,Y) に変化があったことを意味する．34 行目のルールは，各ステップ T において，状態が変化した辺の数が d 以下であることを保証する．

マルチショット符号化． シングルショット符号化は，根付き全域森問題の改良符号化 (コード 3) の自然な拡張になっている．この符号化を用いて遷移問題を解くには，ステップ長 t を増やしながら，複数の問題を繰り返し解く必要がある．しかし，各問題中の制約の大部分は共通であるため，ASP システムが同一の探索空間を何度も調べることになり，求解効率が低下するという問題点がある．

この問題を解決するために，ASP システム *clingo* のマルチショット ASP 解法を適用する．この解法は，ASP システムが同様の探索失敗を避けるために獲得した学習節を (部分的に) 保持することで，無駄な探索を行うことなく，制約を追加した論理プログラムを連続的に解くことができる．そのため，求解性能の向上が期待できる．

ASP システム *clingo* のマルチショット ASP 解法ライブラリを用いた符号化をコード 5 に示す．この符号化は，base，step(t)，check(t) の 3 つのパートから構成される．まず，base パートに，ステップ t=0 で満たすべき制約を記述する．ここでは，スタート状態とステップ 0 の対応を記述する (5 行目)．次に，step(t) パートには，各ステップ t において満たすべき制約を記述する．ここでは，根付き全域森の制約と遷移制約を記述する (11〜37 行目)．最後に，check(t) パートには，プログラムの終了条件を記述する．ここでは，ゴール状態とステップ t の対応を記述する (43 行目)．なお，t がインクリメントされると，一つ前の不要になった終了条件は，query(t) の真偽を動的に操作することにより無効化される．

6 実行実験

提案アプローチの有効性を評価するために，節 4 と節 5 の符号化に基づくソルバーを開発し，実行実験を行った．

根付き全域森問題． ベンチマークとしては，DNET [†5] で公開されている配電網問題 3 問，および，Graph Coloring and its Generalization [†6] で公開されているグラフ彩色問題を基に生成した 82 問 [†7] を使用した．ベンチマーク問題 (計 85 問) の規模は，ノード数 11〜1406，辺の数 16〜49629，根ノード数 1〜281 である．ASP システムには *clingo*-5.4.0 (*trendy*) を使用し，問題 1 問あたりの制限時間は 1 時間とした．実験環境は，Mac mini，3.2 GHz Intel Core i7，64GB メモリである．

基本符号化と改良符号化の比較結果を図 6 に示す．この図はカクタスプロットと呼ばれ，縦軸が CPU 時間，横軸が解けた問題数を表す．グラフが下に寄るほどより高速に，右に寄るほどより多くの問題を解いたことを意味する．図 6 より，改良符号化は，基本符号化と比較して，より多くの問題を高速に解いていることがわかる．

表 1 は，解けた問題数を，ベンチマーク問題に含まれる辺の数で分類したものである．改良符号化は，辺の数が 40,000 を超えるような問題も解けており，大規模な問題に対する有効性が確認できた．

根付き全域森遷移問題． ベンチマークとしては，DNET で公開されている実用規模の配電網問題 (fukui-tepco，ノード数 432，根ノード数 72) をベースにした．この問題の実行可能解から，スタート状態を 5

†5 https://github.com/takemaru/dnet

†6 http://mat.tepper.cmu.edu/COLOR04/

†7 グラフ彩色問題 127 問の中から，連結グラフで辺の数が 50,000 以下である 82 問を使用した．根については全ノードの 1/5 をランダムに選んで使用した．

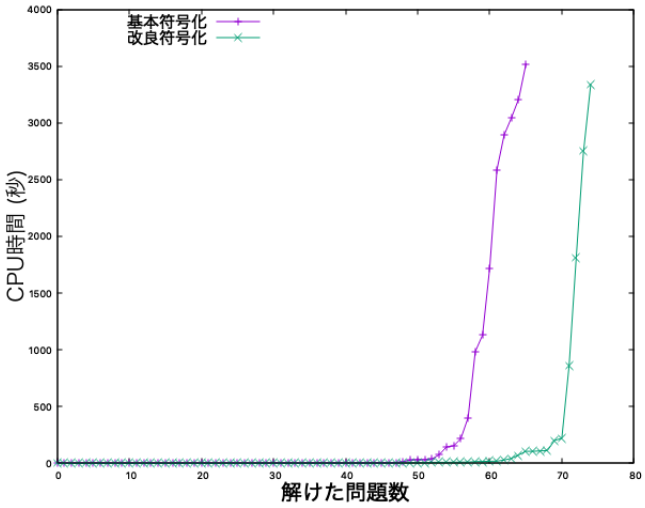

図 6　基本符号化 (コード 2) と改良符号化 (コード 3) の比較

表 1　基本符号化 (コード 2) と改良符号化 (コード 3) の比較 (解けた問題数)

辺の数			問題数	基本符号化	改良符号化
1	～	1000	30	**30**	**30**
1001	～	4000	20	**20**	**20**
4001	～	7000	11	9	**10**
7001	～	10000	8	4	**6**
10001	～	20000	9	2	**5**
20001	～	30000	2	1	**2**
30001	～	40000	1	0	0
40001	～	50000	4	0	**2**
計			85	66	**75**

つ，ゴール状態を 6 つ，をランダムに選び，それらを組み合わせた計 30 問の根付き全域森遷移問題を生成した．ASP システムと実験環境は上と同じである．

シングルショット符号化 (コード 4) とマルチショット符号化 (コード 5) の比較結果を表 2 に示す．左から順に，問題名，解を求めるまでのステップ長，シングルショット符号化とマルチショット符号化のそれぞれの CPU 時間 (秒)，マルチショット符号化を 1 としたときのシングルショット符号化の比率を示している．マルチショット符号化は，シングルショット符号化と比較して，すべての問題をより高速に解いており，平均で 3.3 倍の高速化を実現している．これにより，根付き全域森遷移問題に対するマルチショット ASP 解法の優位性が確認できた．

7　おわりに

本稿では，根付き全域森問題とその解の遷移問題について，解集合プログラミング技術を用いた解法を提案した．根付き全域森問題を解く 2 種類の ASP 符号化を考案した．特に，改良符号化は，根付き全域森問題の連結制約を ASP の個数制約で表現することにより，基礎化後のルール数を少なく抑えるよう工夫

表 2　シングルショット符号化 (コード 4) とマルチショット符号化 (コード 5) の比較結果

問題名	ステップ長 t	シングルショット (秒)	マルチショット (秒)	比率 (シングル/マルチ)
s1_g60	8	50.098	25.659	1.952
s1_g70	8	47.177	24.981	1.889
s1_g80	8	43.193	15.852	2.725
s1_g90	8	48.984	22.983	2.131
s1_g100	6	20.964	4.709	4.452
s10_g60	6	21.947	5.132	4.277
s10_g70	8	43.840	16.290	2.691
s10_g80	6	21.156	4.887	4.329
s10_g90	6	21.276	5.324	3.996
s10_g100	8	45.704	17.697	2.583
s20_g60	6	21.202	4.973	4.263
s20_g70	4	9.890	2.699	3.664
s20_g80	8	48.473	16.335	2.967
s20_g90	10	107.938	64.441	1.675
s20_g100	8	48.473	17.894	2.709
s30_g60	6	21.189	5.287	4.008
s30_g70	4	9.901	2.735	3.620
s30_g80	6	21.884	5.223	4.190
s30_g90	4	9.979	2.658	3.754
s30_g100	8	50.344	18.845	2.671
s40_g60	8	44.637	14.254	3.132
s40_g70	6	21.334	4.795	4.449
s40_g80	8	45.202	14.099	3.206
s40_g90	6	21.710	5.119	4.241
s40_g100	6	21.299	5.666	3.759
s50_g60	4	10.021	2.726	3.676
s50_g70	6	21.291	4.718	4.513
s50_g80	6	21.163	6.503	3.254
s50_g90	10	108.299	65.352	1.657
s50_g100	4	9.934	2.700	3.679
平均比率				3.337

されている．解の遷移問題については，マルチショット ASP 解法を利用して解く方法を提案した．この方法は，ASP システムが同様の探索失敗を避けるために獲得した学習節を (部分的に) 保持することで，無駄な探索を行うことなく，制約を追加した問題を連続的に解くことができる．DNET の問題集，および，Graph Coloring and its Generalizations の問題を元に生成した問題集を用いた実験の結果，大規模な根付き全域森問題に対する改良符号化の有効性，遷移問題に対するマルチショット ASP 解法の優位性が確認

できた．今後の課題としては，配電網問題の電気制約を，背景理論付き ASP (ASP Modulo Theories) を用いて実装することが挙げられる．

参 考 文 献

[1] Baral, C.: *Knowledge Representation, Reasoning and Declarative Problem Solving*, Cambridge University Press, 2003.

[2] Erdem, E., Gelfond, M., and Leone, N.: Applications of ASP, *AI Magazine*, Vol. 37, No. 3(2016), pp. 53–68.

[3] Gelfond, M. and Lifschitz, V.: The Stable Model Semantics for Logic Programming, *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, MIT Press, 1988, pp. 1070–1080.

[4] 林泰弘, 川崎章司, 松木純也, 松田浩明, 酒井重和, 宮崎輝, 小林直樹: 分散型電源連系配電ネットワークの標準解析モデルの構築とネットワーク構成候補の多面的評価手法の開発, 電気学会論文誌, Vol. 126, No. 10(2006), pp. 1013–1022.

[5] 井上克巳, 坂間千秋: 論理プログラミングから解集合プログラミングへ, コンピュータソフトウェア, Vol. 25, No. 3(2008), pp. 20–32.

[6] 井上武, 高野圭司, 渡辺喬之, 川原純, 吉仲亮, 岸本章宏, 津田宏治, 湊真一, 林泰弘: フロンティア法による電力網構成制御, オペレーションズ・リサーチ, Vol. 57, No. 11(2012), pp. 610–615.

[7] 川原純, 湊真一: グラフ列挙索引化技法の種々の問題への適用, オペレーションズ・リサーチ, Vol. 57, No. 11(2012), pp. 604–609.

[8] Niemelä, I.: Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm, *Ann. Mathematics and Artificial Intelligence*, Vol. 25, No. 3–4(1999), pp. 241–273.