情報工学コース卒業研究報告

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考察

2020年2月

山田 健太郎

概要

本論文では，解集合プログラミングを用いた配電網問題の解法について述べる．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．研究対象とす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問題に帰着できることが知られている．

解集合プログラミング (ASP) 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 システムは，一階論理に基づく ASP 言語によって記述された論理プログラムから解集合を計算するシステムである．

本論文では，ASP を用いた根付き全域森探索問題 (SRFP) の解法について述べる．まず，根付き全域森探索問題を解く 2 種類の ASP 符号化 `srf1` と `srf2` を考案した．`srf1` 符号化は，SRFP の根付き連結制約を at-least-one 制約と at-most-one 制約で表現した基本的な符号化に対し，`srf2` 符号化は，根付き連結制約を ASP の個数制約を用いて表現している点が特長である．`srf2` 符号化は，`srf1` 符号化と比較すると基礎化後の制約数を少なく抑えることができるため，大規模な問題に対する有効性が期待できる．

考案した符号化の有効性を評価するために，様々なスイッチ数をもつ 85 問の問題を用いて，実行実験を行なった．その結果，`srf2` 符号化は，`srf1` 符号化より多くの問題を解くことに成功した．また，`srf2` 符号化はスイッチ数が 40,000 個を超えるような大規模な問題も解いており，配電網問題に対する ASP の有効性が確認できた．

目 次

第1章　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 [2, 3]．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 [6]．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．

トポロジ制約のみの配電網問題の例を図 1.1 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 $\{c_1, c_2, c_4\}$ 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 [8]．本研究ではトポロジ制約のみの配電網問題を対象とする．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．以降，この問題を**根付き全域森探索問題** (Spanning Rooted Forest Problem; SRFP) と呼ぶ．

解集合プログラミング (Answer Set Programing; ASP[1, 5, 10, 7]) は，

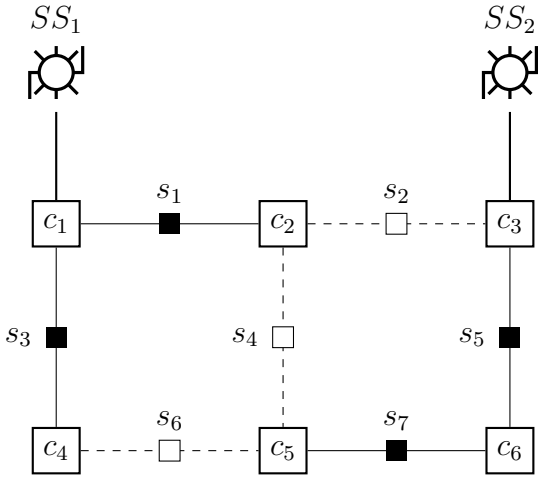


図 1.1: 配電網問題の例

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 [5] 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 [4].

本論文では，解集合プログラミング (ASP) を用いた根付き全域森探索問題 (SRFP) の解法について述べる．まず，根付き全域森探索問題を解く 2 種類の ASP 符号化 `srf1` と `srf2` を考案した．これらの符号化は，SRFP の制約を 7 つまたは 6 つの ASP のルールで簡潔に表現している．`srf1` 符号化は，SRFP の根付き連結制約を at-least-one 制約と at-most-one 制約で表現した基本的な符号化に対し，`srf2` 符号化は，根付き連結制約を ASP の個数制約を用いて表現している点が特長である．`srf2` 符号化は，`srf1` 符号化と比較すると基礎化後の制約数を少なく抑えることができるため，大規模な問題に対する有効性が期待できる．

また，障害時の復旧予測への応用を狙いとした，ある初期配電網の構成からスタートして，トポロジ制約を満たした上で，一度に切り替えるスイッチの数を k 個以下に制限し，最終的に目的とする配電網の構成を得るためのスイッチの切り替え手順を求める遷移問題への拡張も行った．

考案した符号化の有効性を評価するために，DNET (Power Distribu-

tion Network Evaluation Tool) [1] に公開されている問題 (3 問) と，Graph Coloring and its Generalizations [2] に公開されているグラフ問題を元に生成した問題 (82 問) を用いて，実行実験を行なった．その結果，srf2 符号化は，srf1 符号化より多くの問題を解くことに成功した．また，srf2 符号化はスイッチ数が 40,000 個を超えるような大規模な問題も解いており，配電網問題に対する ASP の有効性が確認できた．

本論文の構成は，第 2 章で根付き全域森の定義を示し，第 3 章で解集合プログラミングの説明を行う．第 4 章で根付き全域森問題の ASP 符号化と考案したプログラムを示し，第 5 章で考案したプログラムの評価実験とその考察を述べる．第 6 章で根付き全域森問題の遷移問題への拡張を行い，第 7 章で本稿についてまとめる．

[1] https://github.com/takemaru/dnet
[2] https://mat.tepper.cmu.edu/COLOR04/

第2章　配電網と根付き全域森

先に述べた通り，トポロジ制約のみの配電網問題は変電所を根とする根付き全域森探索問題に帰着される．本章では，扱う配電網のグラフ表現と根付き全域森の定義について説明を行う．

2.1　配電網のグラフ表現

配電網 (図 1.1) を表すグラフ表現を図 2.1 に示す．なお，スイッチの開閉状態は考慮しないものとする．

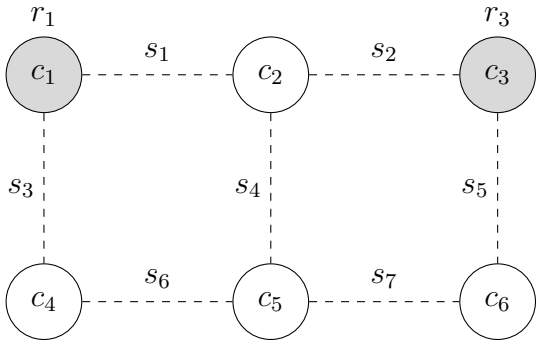


図 2.1: 図 1.1 を表すグラフ表現

図 1.1 における需要家 $\{c_1, \ldots, c_6\}$ は，グラフの**ノード**に対応する．スイッチ $\{s_1, \ldots, s_7\}$ は，グラフの**辺**に対応し，無向辺として表現する．図中の色付きノード $\{c_1, c_3\}$ は変電所と直接繋がっている需要家を意味しており，**根**を表している．また，根はノードの添字を用いて $\{r_1, r_3\}$ のように表す．以降では，このグラフ表現によって問題の説明を行う．

2.2　根付き全域森

構成する配電網により，電力は変電所から全ての需要家のもとへ届けられなければならない．スイッチの状態によって決まる供給経路は，短絡の発生を防ぐために変電所を唯一の根とする木となる．また，配電網全体を複数の木で覆うことで，停電が発生しないようにする．つまり，トポロジ制約を満たす配電網問題の解は根付き木の集合である根付き全域森となる．

根付き全域森は以下のように定義される．[9]

定義　グラフ $G = (V, E)$ と根 R が入力として与えられたとする．このとき，G 上の根付き全域森とは，以下の 2 つの制約を満たす 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**)

図 2.1 における根付き全域森の例を図 2.2 に示す．

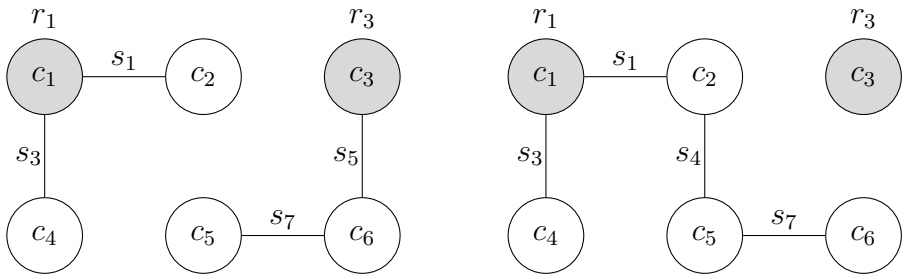


図 2.2: 根付き全域森の例

図 2.2 のように根付き全域森は 1 つの入力グラフに対し，複数存在する．根付き全域森は，非連結であることを許すが，各連結成分が必ずちょうど 1 つの根を持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 2.2(左) は，図 1.1 で表した実行可能解をグラフ表現に置き換えたものである．また，解の一部には図 2.2(右) のように，ある連結成分が根のみの場合もある．

第3章　解集合プログラミング

本章では，解集合プログラミングについて述べる．解集合プログラミング (ASP; [1, 5, 10, 7]) の言語は，一般拡張選言プログラムをベースとしている．本論文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．

$$a_0 \texttt{ :- } a_1\texttt{,}\ldots\texttt{,}a_m\texttt{,not } a_{m+1}\texttt{,}\ldots\texttt{,not } a_n\texttt{.} \tag{3.1}$$

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**[1]，“`,`” 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，“`:-`” を省略してよい．

ヘッドが空のルールを**一貫性制約**と呼ぶ．

$$\texttt{:- } a_1\texttt{,}\ldots\texttt{,}a_m\texttt{,not } a_{m+1}\texttt{,}\ldots\texttt{,not } a_n\texttt{.} \tag{3.2}$$

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 “$\{a_1\ ;\ \ldots\ ;\ a_n\}$`.`” 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，“$lb\,\{a_1\ ;\ \ldots\ ;\ a_n\}\,ub$ `:-` $Body$`.`” と書くと，「$Body$ 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

[1] **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 ($\neg$) とは意味が異なる．

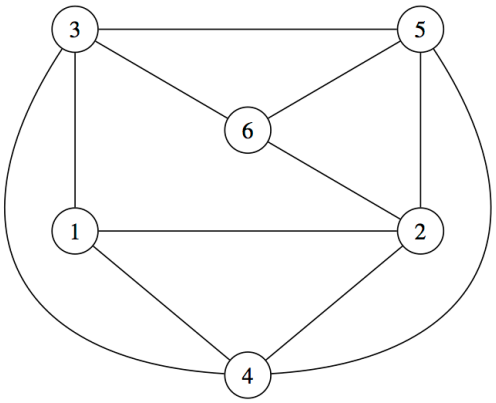


図 3.1: グラフ

れている．例えば，“`:- not` lb `#sum`$\{1{:}a\ ;\ 2{:}b\ ;\ 3{:}c\}\ ub$.” と書くと，「$a$ が成り立つとき重み 1，b が成り立つとき重み 2，c が成り立つとき重み 3 とし，その和が lb 以上 ub 以下になるように部分集合が成り立つ」を意味する．また，組合せ最適化問題を解くために，最小化関数 (`#minimize`)・最大化関数 (`#maximize`) 等も用意されている．

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 [5] に基づく解集合を計算するシステムである．近年，*clingo* [2]，*DLV* [3]，*WASP* [4] など，SAT ソルバー技術を応用した高速な ASP システムが開発されている．なかでも *clingo* 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．

解集合プログラミングを用いた問題解法プロセスは，3 つのステップからなる．まず最初に，解きたい問題を論理プログラムとして表現する．つぎに，ASP システムを用いて，論理プログラムの解集合を計算する．最後に，解集合を解釈して元の問題の解を得る．ここでは，グラフ彩色問題を例として，各ステップ毎に解法プロセスを説明する．ASP システムとしては *clingo* を用いる．

グラフ彩色問題とは，辺で結ばれたノードが同じ色にならないように，各ノードを塗り分ける問題である．例として，図 3.1 のグラフを赤 (`r`)，青 (`b`)，緑 (`g`) の 3 色で塗り分ける問題を考える．この問題を表す論理プログラムをコード 3.1 に示す．

---

[2] `https://potassco.org/`
[3] `http://www.dlvsystem.com/dlv/`
[4] `https://www.mat.unical.it/ricca/wasp/`

```
1  % グラフ
2  node(1).    node(2).    node(3).   node(4).    node(5).    node(6).
3  edge(1,2).  edge(1,3).  edge(1,4). edge(2,4).  edge(2,5).  edge(2,6).
4  edge(3,4).  edge(3,5).  edge(3,6). edge(4,5).  edge(5,6).
5
6  % 色
7  col(r).   col(b).   col(g).
8
9  % 制約
10 1 { color(X,C) : col(C) } 1 :- node(X).
11 :- edge(X,Y), color(X,C), color(Y,C).
12
13 #show color/2.
```

コード 3.1: グラフ彩色問題の論理プログラム (color.lp)

```
clingo version 5.3.0
Reading from color.lp
Solving...
Answer: 1
color(2,b) color(1,g) color(3,b) color(4,r) color(5,g) color(6,r)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.004s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.004s
```

コード 3.2: color.lpに対する *clingo* の実行例

2〜4行目は，ノード (node) と辺 (edge) をファクトとして書くことによって，図 3.1 のグラフを表している．7行目も同じく，色 (col) をファクトで表している．10行目のルールは，個数制約を使って「各ノードは一つの色で塗られる」という制約を表している．アトム color(X,C) は，ノード X が色 C で塗られることを意味する．“:” は条件付きリテラルと呼ばれる拡張構文であり，このルールのヘッドは，1 {color(X,r);color(X,b);color(X,g)} 1 のように展開される．11 行目のルールは，一貫性制約を使って 「辺で結ばれたノード (X と Y) は，同じ色 (C) で塗られない」という制約を表している．

ASP システムは解集合を計算して出力する．コード 3.2 に *clingo* の実行例を示す．この出力から，ノード 1 と 5 は緑，ノード 4 と 6 は赤，ノード 2 と 3 は青に塗り分けられることがわかる．

第4章　根付き全域森のASP符号化

本章では，解集合プログラミングを用いて根付き全域森を論理プログラムで表現する方法について述べる．

4.1　ASPファクト形式

入力として与えられるグラフはASPのファクトで表される．図2.1をファクト形式で表現したものをコード4.1に示す．

```
1 %% node
2 node(1..6).
3
4 %% root
5 root(1).  root(3).
6
7 %% edge
8 edge(1,2).  edge(1,4). edge(2,5).  edge(2,3).
9 edge(3,6).  edge(4,5). edge(5,6).
```

コード 4.1: 図2.1のファクト形式 (graph.lp)

nodeは各ノードを表しており，2行目で添字が1から6までのノードを定義している．アトムroot(X)はノードXが根であることを意味する．5行目で1と3のノードが根であることを定義している．アトムedge(X,Y)はノードXとYが辺で結ばれていることを意味する．8〜9行目で7つの各辺を定義している．例として，図2.1における辺s_1は，edge(1,2)のように表される．

```
1  %% choose edge
2  { inForest(X,Y) } :- edge(X,Y).
3
4  %% generate reached
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  %% non-cycle constraint
10 :- root(R), not 1 #sum{ 1,X:reached(X,R) ; -1,X,Y:inForest(X,Y),reached(X,R),reached(Y,
       R)} 1.
11
12 %% rooted connection constraint
13 :- node(X),  not reached(X,_).
14 :- reached(X,R1), reached(X,R2), R1 < R2.
```

コード 4.2: 根付き全域木の論理プログラム (srf1.lp)

```
1  %% choose edge
2  { inForest(X,Y) } :- edge(X,Y).
3
4  %% generate reached
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  %% non-cycle constraint
10 :- root(R), not 1 #sum{ 1,X:reached(X,R) ; -1,X,Y:inForest(X,Y),reached(X,R),reached(Y,
       R)} 1.
11
12 %% rooted connection constraint
13 :- node(X), not 1 { reached(X,R) } 1.
```

コード 4.3: 根付き全域木の論理プログラム (srf2.lp)

4.2　ASP 符号化

根付き全域森の各種制約は複数のルールの集合として簡潔に記述できる．本研究では，与えられたグラフから根付き全域森探索問題を解く 2 種類の論理プログラムを考案した．考案した srf1 符号化をコード 4.2 に，srf2 符号化をコード 4.3 にそれぞれ示す．

srf1 符号化は，全部で 7 つのルールで根付き全域森の制約を表している．それぞれのルールの意味について説明していく．

2 行目のルールは，選択子を使って「各辺は解となる森に含まれるか含まれないかのどちらかである」という制約を表している．アトム inForest(X,Y) は，辺 edge(X,Y) が解となる森に含まれることを意味する．

5〜7 行目は，各連結成分のノードの集合を生成するルールである．ア

トム reached(X,R)，はノード X が根 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，「各根は自分自身から到達可能である」という制約を表している．6〜7 行目のルールは，「すでに到達可能である各ノードについて，そのノードと辺で結ばれたノードは同じ根から到達可能である」という制約を表している．

10 行目は，根付き全域森の非閉路制約を表すルールである．各連結成分は木であることから，木の辺の数に関する性質[1]を基にしている．10 行目のルールは，重み付き個数制約を使った一貫性制約で 「各根について，(到達可能なノードの数) – (そのノード間で結ばれる辺の数) = 1 である」という制約を表している．

13〜14 行目は，根付き全域森の根付き連結制約を表すルールである．13 行目のルールは，一貫性制約を使って 「各ノードは少なくとも 1 つの根から到達可能である」という at-least-one 制約を表している．14 行目のルールは，一貫性制約を使って 「各ノードは高々1 つの根から到達可能である」という at-most-one 制約を表している．この 2 つの制約が各根に対する到達可能な根の数の上下限を表すことで根付き連結制約を表している．

srf2 符号化は，全部で 6 つのルールで根付き全域森の制約を表している．2〜10 行目までの各ルールは srf1 符号化と同様にして制約を表している．

13 行目は，根付き連結制約を表すルールである．ASP の拡張構文である個数制約を使った一貫性制約で 「各ノードはちょうど 1 つの根から到達可能である」ことを直接表している．

根付き連結制約の表現による違いについて，srf2 符号化は srf1 符号化と比べて，問題に符号化を適用した際に生成される制約数を少なく抑えられることが特長である．具体的には，ノードの数が n 個，根の数が r 個の問題について，各符号化の根付き連結制約を表すルールが生成する制約数を確認する．srf1 符号化では，13 行目のルールが各ノードに対して符号化した n 個と，14 行目のルールで，各ノードについて 2 つの根を選択することにより符号化した $n \times {}_rC_2$ 個であり，全部で $n(1 + {}_rC_2)$ 個である．一方，srf2 符号化は，13 行目のルールで各ノードに対して符号化するので，全部で n 個と少なくなっていることがわかる．

[1] ノード数 n の木について，辺の数は $n - 1$ となる．

第5章　評価実験

本章では，ベンチマーク問題を使用した実験結果を通じて提案する2種類のASP符号化の比較評価を行う．

ベンチマーク問題として，DNET[1] で公開されている配電網モデル3問と，Graph Coloring and its Generalization[2] で公開されているグラフ彩色問題127問中，辺の数が50000以下である非連結成分を含まない無向グラフ82問に対し，ノードのうち1/5個をランダムに根として与えたものを用いた．これらの計85問の問題のノードの数は11〜1406，辺の数は16〜49629，根の数は1〜281である．

実験は，提案した2種類のASP符号化`srf1`(コード 4.2)，`srf2`(コード 4.3) を用いて符号化した170問について，それぞれ制限時間を1時間として *clingo* で実行し，解いた問題数，各問題の実行CPU時間と制約数を集計する．なお，*clingo* のバージョンは5.4.0，オプションとして *trendy* を使用した．実験環境は，Mac mini，3.2 GHz Intel Core i7，64GBメモリである．

はじめに，各符号化の性能を比較するためにカクタスプロットによる比較結果を図5.1に示す．カクタスプロットは，縦軸が各問題を解くまでのCPU時間を表し，横軸が解けた問題数を表す．グラフが右に寄るほど制限時間内に多くのベンチマーク問題を解くことが可能であることを意味するので，符号化の性能が良いとされる．

結果では，`srf2`符号化は`srf1`符号化よりも解いた問題数が多いことを示した．また，今回実験を行った多くの問題に対してASPによる全域森探索は実行CPU時間が小さいことを示した．

次に，実際に各符号化が解いた問題について，問題の規模ごとに分類して比較したものを表5.1に示す．表中の太字は，解けた問題数を比較して多い方，または同数であることを表している．

結果から，辺の数が少ない問題では符号化による解けた問題数の差は大

[1] https://github.com/takemaru/dnet
[2] https://mat.tepper.cmu.edu/COLOR04/

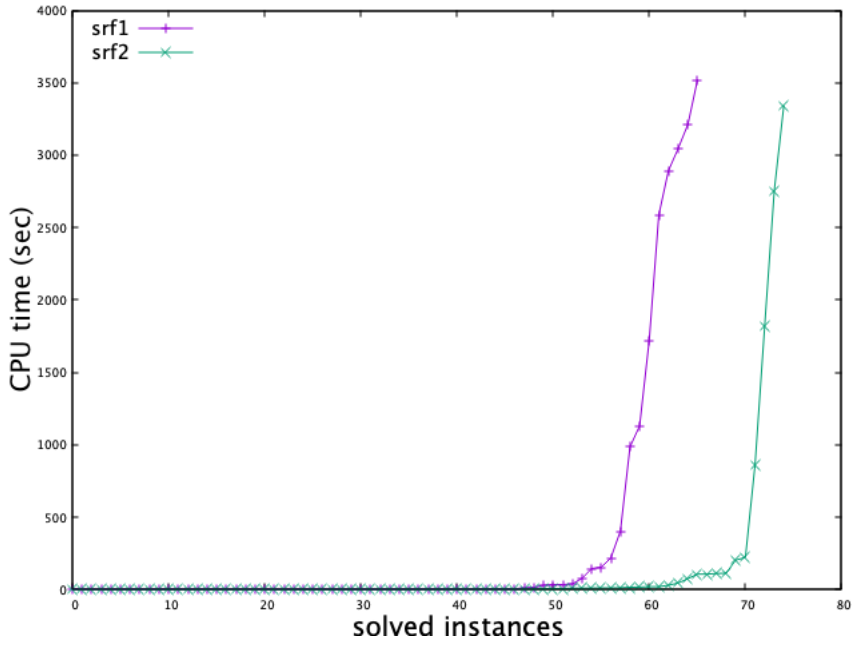


図 5.1: 比較結果: カクタスプロット

表 5.1: 比較結果: 解けた問題数 (問)

辺の数	問題数	srf1	srf2
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**

きく見られないが，辺の数が 10,000 を超える問題では，srf2 の方が srf1 よりも多くの問題を解いたことを示した．また，srf2 による符号化では，辺の数が 40,000 を超えるような大規模の問題に対して，解を求めることに成功した．これらの結果から，4 章で述べた srf2 符号化の特長である，根付き連結制約を符号化した際の制約数が少なく抑えられることが拡張性の高さに有効であったといえる．

実際に srf2 符号化の方が制約数が少ないことを確認するために，辺の数が 7000 より大きい問題について，生成された制約数の比較結果を図 5.2 に示す．

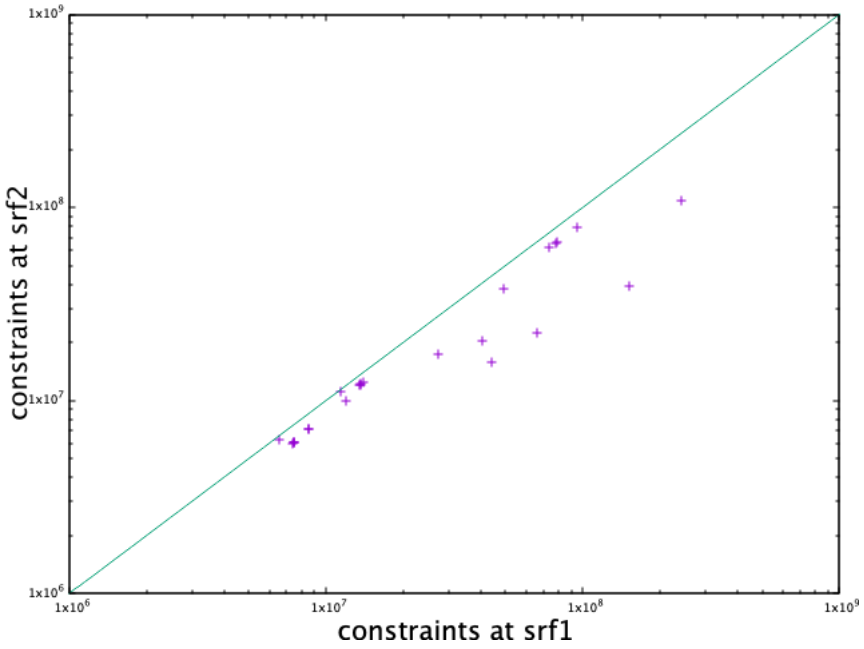


図 5.2: 比較結果: 生成された制約数

図 5.2 は，横軸が srf1 による制約数，縦軸が srf2 による制約数を示す．図中のラインは，2 つの符号化の生成される制約数が同じになる境界を表している．記号 + は問題を表しており，その座標が各符号化の制約数となる．記号が境界から下にあれば，その問題では srf1 の方が制約数が少なくなることを意味する．反対に，記号が境界から上にあれば，srf2 の方が制約数が少なくなることを意味する．

結果を見ると，記号は全体的に境界よりも下にあるため，確かに srf2 による符号化は生成される制約の数が少なく抑えられていることが確認できる．

第6章　遷移問題への拡張

本章では，配電網問題を遷移問題として解くことを想定し，根付き全域森を遷移問題に拡張する手法について説明する．また，簡単な実行例も示す．

6.1　遷移問題の概要

配電網問題を遷移問題へ拡張することで，障害時の復旧予測や，現在の電力網構成から目的の構成へ切り替えるための手順探索などに応用できる．しかし，スイッチの開閉を切り替える際に電力の供給が停止してしまうことを可能な限り防ぐ必要がある．また，一度に多くのスイッチを切り替えることは困難であると考えられる．そのため，本研究では，初期の配電網構成から，トポロジ制約を満たした上で，一度に切り替えられるスイッチの個数に制限を設けて，目的の配電網構成を得る遷移問題を想定した根付き全域森探索問題の拡張を行う．

根付き全域森の遷移問題への拡張について，新たに**初期状態**と**目的状態**を定義する．初期状態と目的状態は，それぞれ対象となるグラフの根付き全域森の 1 つの解である．

根付き全域森の遷移問題は，入力として初期状態と目的状態が与えられ，各ステップで根付き全域森の制約を満たしながらその状態遷移を求める問題である．また，本研究において遷移問題が満たすべき新たな 2 つの制約を以下に示す．

1. 初期状態から目的状態までの遷移にかかるステップ数は t 以下である．(**ステップ数制約**)

2. 状態が遷移する際に，変化する辺の数は k 個以下である．(**遷移制約**)

また，ASP 言語の拡張性を利用することで，あるステップにおいては特定の辺が使えないなどの様々な制約を簡潔にルールとして表現することが可能であり，柔軟に追加することもできる．

```
1  %% constant
2  #const t = 3.
3  #const k = 2.
4  t(0..t).
5
6  %% init
7  :- not inForest(X,Y,0), init_Forest(X,Y).
8
9  %% goal
10 :- not inForest(X,Y,t), goal_Forest(X,Y).
11
12 %% choose switch
13 { inForest(X,Y,T) } :- edge(X,Y), t(T).
14
15 %% generate reached
16 reached(R,R,T) :- root(R), t(T).
17 reached(X,R,T) :- reached(Y,R,T), inForest(Y,X,T), t(T).
18 reached(X,R,T) :- reached(Y,R,T), inForest(X,Y,T), t(T).
19
20 %% non-cycle constraint
21 :- 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.
22
23 %% rooted connection constraint
24 :- node(X), t(T), not 1 { reached(X,R,T) } 1.
25
26 %% transition constraint
27 dist(X,Y,T) :- inForest(X,Y,T), not inForest(X,Y,T-1), t(T), T>0.
28 dist(X,Y,T) :- inForest(X,Y,T-1), not inForest(X,Y,T), t(T), T>0.
29 :- t(T), not #sum{ 1,X,Y:dist(X,Y,T) } k.
```

コード 6.1: 遷移問題の論理プログラム (srf-transition.lp)

6.2　ASP 符号化

遷移問題に拡張した符号化 srf-transition をコード 6.1 に示す．問題の拡張にあたり，根付き全域森の符号化は srf2(コード 4.3) を基にした．遷移問題の符号化は，全部で 11 個のルールで表現される．

2〜4 行目は定数であり，定数 t はステップ数制約の最大ステップ数を表す．定数 k は遷移制約において，最大で変化することのできる辺の数を表す．4 行目のファクト t(0..t). は，0 から t までの各ステップ数を定義している．

7 行目と 10 行目は，初期状態と目的状態に関する制約である．アトム init_Forest(X,Y) は，初期状態での根付き全域森を意味する．アトム goal_Forest(X,Y) は，目的状態での根付き全域森を意味する．7 行目のルールは，「初期状態と 0 ステップ目の根付き全域森は一致する」という制約を表している．10 行目のルールは，「最終状態と t ステップ目の根付き全域森は一致する」という制約を表している．この 10 行目の制約を満

たすことで，遷移問題のステップ数制約も満たすことを意味する．

13，16〜18，21，24 行目は 4 章で説明した各ルールに，ステップ数を意味する項 T を導入し，各ステップにおいて各アトムが成り立つように拡張する．これにより，各ステップにおける根付き全域森の制約を表している．

27〜29 行目は遷移制約を表すためのルールである．アトム dist(X,Y,T) は，T ステップ目で辺 edge(X,Y) が変化したことを意味する．27 行目のルールは，「T>0 となる各ステップについて，T-1 ステップ目で森に含まれていない辺が，T ステップ目で含まれているならば，変化した辺である」という制約を表している．28 行目のルールは，逆に 「T>0 となる各ステップについて，T-1 ステップ目で森に含まれている辺が，T ステップ目で含まれていないならば，変化した辺である」という制約を表している．29 行目のルールは，「各ステップについて，変化した辺の数は k 以下となる」という遷移制約を表している．個数制約を使うことで T ステップ目のアトム dist(X,Y,T) の数が k 以下になることを意味する．

6.3 実行例

入力として与える初期状態を図 6.1 に，目的状態を図 6.2 にそれぞれ示す．また，ASP ファクト形式で表したものをコード 6.2 に示す．

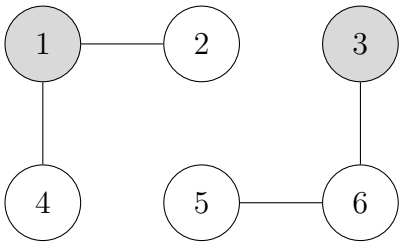


図 6.1: 初期状態

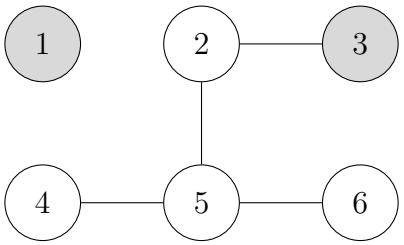


図 6.2: 目的状態

実行例として，この入力に対してコード 6.1 の論理プログラムを $t = 3$ のステップ数制約と $k = 2$ の遷移制約として，*clingo* で実行した結果をコード 6.3 に示す．また，この出力をグラフ表現に変換したものを図 6.3 に示す．

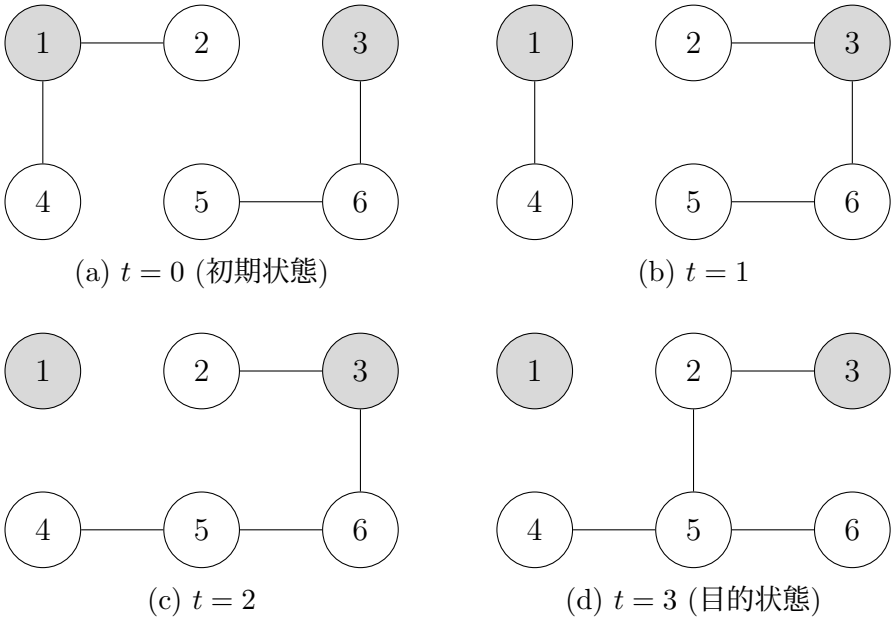
```
1  %% node
2  node(1..6).
3
4  %% root
5  root(1). root(3).
6
7  %% edge
8  edge(1,2). edge(1,4). edge(2,3).
9  edge(2,5). edge(3,6). edge(4,5). edge(5,6).
10
11 % init
12 init_Forest(1,2). init_Forest(1,4).
13 init_Forest(3,6). init_Forest(5,6).
14
15 % goal
16 goal_Forest(2,3). goal_Forest(2,5).
17 goal_Forest(4,5). goal_Forest(5,6).
```

コード 6.2: 入力のファクト形式 (graph-transition.lp)

```
clingo version 5.4.0
Reading from graph-transition.lp ...
Solving...
Answer: 1
t(0) t(1) t(2) t(3) init_Forest(1,2) init_Forest(1,4) init_Forest(3,6) init_Forest(5,6)
      goal_Forest(2,3) goal_Forest(2,5) goal_Forest(4,5) goal_Forest(5,6) edge(1,2)
     edge(1,4) edge(2,3) edge(2,5) edge(3,6) edge(4,5) edge(5,6) root(1) root(3)
     reached(1,1,0) reached(1,1,1) reached(1,1,2) reached(1,1,3) reached(3,3,0) reached
     (3,3,1) reached(3,3,2) reached(3,3,3) node(1) node(2) node(3) node(4) node(5) node
     (6) reached(2,3,3) reached(6,3,3) reached(6,3,0) reached(2,1,0) reached(4,1,0)
     reached(5,3,0) reached(5,3,3) reached(4,3,3) inForest(1,2,0) inForest(1,4,0)
     inForest(1,4,1) inForest(2,3,1) inForest(2,3,2) inForest(2,3,3) inForest(2,5,3)
     inForest(3,6,0) inForest(3,6,1) inForest(3,6,2) inForest(4,5,2) inForest(4,5,3)
     inForest(5,6,0) inForest(5,6,1) inForest(5,6,2) inForest(5,6,3) dist(1,2,1) dist
     (2,3,1) dist(1,4,2) dist(4,5,2) dist(2,5,3) dist(3,6,3) reached(2,3,2) reached
     (2,3,1) reached(6,3,2) reached(6,3,1) reached(4,1,1) reached(5,3,1) reached(5,3,2)
      reached(4,3,2)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.010s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.016s
```

コード 6.3: graph-transition.lp に対する *clingo* の実行例



(a) $t = 0$ (初期状態) (b) $t = 1$

(c) $t = 2$ (d) $t = 3$ (目的状態)

図 6.3: 実行例 (コード 6.3) のグラフ表現

第7章　結論

本研究では，ASP を用いた根付き全域森探索問題の解法について述べた．考案した 2 種類の ASP 符号化 (srf1,srf2) は，根付き全域森の制約を ASP 言語の高い表現力によって数行で表現することができた．特に，srf2 符号化の特長である，基礎化後の制約数を少なく抑えるという点は，評価実験において，srf1 符号化よりも多くの問題を解く結果を示した．また，srf2 符号化は，辺の数が 40,000 を超える大規模な問題に対して解を求められたという高い拡張性も示した．さらには，今回行った評価実験で使った問題の多くを高速に解いており，配電網問題に対する ASP の有効性が確認できた．

遷移問題への拡張については，設定した新たな制約を追加しても，ASP 言語の高い表現力により，5 つのルールを追加することで問題を解くことが可能であることが確認できた．

今後の課題として，今回の研究では遷移問題の評価実験を行うことが出来なかったので，それについて行いたい．さらなる遷移問題の応用として，様々な制約を追加することが可能であるので，問題背景から考えられる制約の設定を追加し，その評価を行いたい．

また，トポロジ制約のみを対象とした配電網問題においては，ASP の有効性が確認できた．しかし，電力制約においては複雑な実数の計算を扱う必要があり，ASP ソルバーだけでは実数による制約を扱うのが難しい．そのため，トポロジ制約のみを ASP ソルバーで解き，その解について背景理論ソルバーを適用し，最適化探索を行うなどの実装方法について考えたい．

謝辞

本研究を行うにあたり，手厚いご指導を頂いた指導教官の番原 睦則 教授に深く感謝申し上げます．また，実験に使わせて頂いた配電網問題を公開なされているNTT 未来ネット研究所の井上 武 様に感謝申し上げます．研究を通じて，様々な相談に乗って頂いた番原研究室の皆様に感謝申し上げます．最後に，これまで支えて頂いた両親や友人に感謝申し上げます．

参考文献

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

[2] M. E. Baran and F. F. Wu. Network reconfiguration in distribution systems for loss reduction and load balancing. *IEEE Transactions on Power Delivery*, Vol. 4, No. 2, pp. 1401–1407, April 1989.

[3] H. . Chiang and R. Jean-Jumeau. Optimal network reconfigurations in distribution systems. i. a new formulation and a solution methodology. *IEEE Transactions on Power Delivery*, Vol. 5, No. 4, pp. 1902–1909, Oct 1990.

[4] E. Erdem, M. Gelfond, and N. Leone. Applications of asp. *AI Magazine*, Vol. 37, No. 3, pp. 53–68, 2016.

[5] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pp. 1070–1080. MIT Press, 1988.

[6] 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, pp. 1013–1022, oct 2006.

[7] 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, pp. 20–32, 2008.

[8] 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, pp. 610–615, nov 2012.

[9] 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, pp. 604–609, nov 2012.

[10] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Mathematics and Artificial Intelligence*, Vol. 25, No. 3–4, pp. 241–273, 1999.