コンピュータ科学科
情報システム系
卒業論文

独立集合遷移問題を解くASP符号化の改良

2025年2月

102130265　平澤 歩武

概要

独立集合遷移問題とは，基となる独立集合問題とその 2 つの実行可能解が与えられたとき，一方から他方へ，遷移制約を満たしつつ実行可能解のみを経由して到達可能かを判定する問題である．この問題は最も広く研究されている組合せ遷移問題の 1 つであり，一般的に PSPACE 完全であることが知られている．また，国際組合せ遷移競技会 CoRe Challenge 2022–2023 のベンチマークとして使用されるなど，その実践的な解法アルゴリズムの研究が盛んに行われている．本論文では，解集合プログラミング (ASP) を用いて TJ 型独立集合遷移問題を解く新たな ASP 符号化を提案する．提案符号化は，TJ 型独立集合遷移問題の制約を 6 個の ASP ルールで簡潔に表現できる．CoRe Challenge の複数部門で優勝した既存符号化との大きな違いは遷移制約の表現である．既存符号化は各遷移においてどの頂点が独立集合に入るかという点に着目する．それに対して，提案符号化は各遷移においてどのトークンが移動するかという点に着目する点が異なる．提案符号化の有効性を評価するため，CoRe Challenge 2022 問題集 (全 369 問) を用いた実験を行った．その結果，提案符号化は，既存符号化と比較して，6 問多い 240 問を解き，その有効性を確認できた．

目 次

図 目 次

表 目 次

コード目次

第1章　序論

組合せ遷移問題 (Combinatorial Reconfiguration Problems [15, 24, 27]) とは，基となる組合せ問題とその 2 つの実行可能解が与えられたとき，一方の実行可能解から他方の実行可能解へ，遷移制約を満たしつつ，実行可能解のみを経由して到達できるかを判定する問題である．組合せ遷移問題の研究は，ここ 20 年で理論計算機科学の分野を中心として急速に発展しており，理論的な基盤が整備されている．例えば，計算複雑性の分野では，組合せ遷移問題の基となる組合せ問題が NP 完全であるとき，その遷移問題の多くは PSPACE 完全であることが示されている [15]．代表的な問題としては，命題論理の充足可能性判定（SAT）遷移 [9, 22]，独立集合遷移 [16, 19]，支配集合遷移 [2, 10, 11, 25]，グラフ彩色遷移 [3, 4, 5]，クリーク遷移 [18]，ハミルトン閉路遷移 [26]，集合被覆遷移 [16] などがあげられる．また，ここ数年では国際組合せ遷移競技会 CoRe Challenge 2022–2023 が開催されるなど，組合せ遷移問題を解くための実践的なアルゴリズムやソルバー開発が盛んに行われている．

独立集合遷移問題 (Independent Set Reconfiguration Problems [15]) は，独立集合問題を基とする組合せ遷移問題である．この問題は，組合せ遷移コミュニティで最も活発に研究されている組合せ遷移問題であり，CoRe Challenge のベンチマークにも採用されている．独立集合遷移問題の遷移制約としては，トークンジャンピング（Token Jumping; TJ）とトークンスライディング（Token Sliding; TS) が最も広く研究されている．独立集合に含まれる頂点にトークンが置かれていると仮定したとき，TJ 型遷移制約は，1 回の遷移でちょうど 1 つのトークンが他の頂点へ移動することを表す．TS 型遷移制約は，1 回の遷移でちょうど 1 つのトークンが隣接頂点へ移動することを表す．本論文では，TJ 型遷移制約をもつ独立集合遷移問題 (以降，TJ 型独立集合遷移問題) を対象とする．

解集合プログラミング (Answer Set Programming; ASP [1, 8, 14, 23]) は，論理プログラミングから派生した比較的新しい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いた知識表現言語の一種であ

る．論理プログラムはルールの有限集合で表される．ASP システムは安定モデル意味論 [8] に基づく解集合を計算するシステムである．2000 年以降，SAT ソルバーの技術を応用した高速 ASP システムが開発され，システム生物学，プランニング，スケジューリング，有界モデル検査など，幅広い分野への実用的応用が活発に行われている [7]．ごく最近では，組合せ遷移問題にも積極的に応用されている [13, 20, 28, 29]．特に，山田らの開発した TJ 型独立集合遷移問題の ASP 符号化，および，ASP に基づく組合せ遷移ソルバー *recongo* は，CoRe Challenge の複数部門で優勝するなど良い性能を示している [28, 29]．

本論文では，解集合プログラミングを用いて TJ 型独立集合遷移問題を解く新たな ASP 符号化を提案する．提案符号化は，TJ 型独立集合遷移問題の制約を 6 個の ASP ルールで簡潔に表現できる．CoRe Challenge の複数部門で優勝した既存符号化 [28, 29] との大きな違いは遷移制約の表現である．既存符号化は各遷移においてどの頂点が独立集合に入るかという点に着目する．それに対して，提案符号化は各遷移においてどのトークンが移動するかという点に着目する点が異なる．提案符号化の有効性を評価するため，CoRe Challenge 2022 問題集 (全 369 問) を用いた実験を行った．その結果，提案符号化は，CoRe Challenge の複数部門で優勝した既存符号化と比較して，6 問多い 240 問を解き，その有効性を確認できた．

本論文の構成は，2 章で組合せ遷移問題と独立集合遷移問題の定義や問題例について述べ，3 章で解集合プログラミングについて説明する．4 章では TJ 型独立集合遷移問題を解く ASP 符号化ついて説明し，5 章で性能評価実験についての説明と考察を行う．また 6 章で関連研究について述べる．最後に 7 章で本論文について総括する．

第2章　独立集合遷移問題

2.1　組合せ遷移問題

組合せ遷移問題 (Combinatorial Reconfiguration Problems [15, 24, 27]) とは，基となる組合せ問題とその 2 つの実行可能解 X_s と X_g，および遷移制約が与えられたとき，遷移系列

$$X_s = X_0 \to X_1 \to \cdots \to X_{\ell-1} \to X_\ell = X_g \tag{2.1}$$

が存在するかどうかを判定する問題である．X_s，X_g はそれぞれスタート状態，ゴール状態を表す．各状態 X_i は基となる組合せ問題の実行可能解を表す．$X \to X'$ は遷移制約を満たしながら状態 X から状態 X' へ 1 回遷移することを表す．ℓ は遷移系列の長さ，すなわち，遷移長を表す．(2.1) のような遷移系列が存在するとき，その組合せ遷移問題は到達可能であるといい，存在しないとき，到達不能であるという．

2.2　独立集合遷移問題

無向グラフ $G = (V, E)$ の頂点の部分集合 $S \subseteq V$ に対して $|S| = k$ であり，任意の 2 頂点 $v_1, v_2 \in S$ に対して $(v_1, v_2) \notin E$ であるとき，S はサイズ k の**独立集合** (Independent Set) である．サイズ k の**独立集合問題**とは，与えられた無向グラフ G と自然数 k に対し，G がサイズ k の独立集合を持つかどうかを判定する問題である．例として，図 2.1 に示したグラフが与えられた場合のサイズ 3 の独立集合問題を解く．ここで，頂点に書かれた数字は頂点番号を表す．この独立集合問題の実行可能解の 1 つを図 2.2 に示す．黄色く色付けされた頂点番号 3，5，7 の頂点は独立集合に含まれることを表しており，それぞれの頂点間に辺が存在しないことがわかる．

独立集合遷移問題 (Independent Set Reconfiguration Problems [15]) とは，独立集合問題を基とする組合せ遷移問題である．すなわち，遷移系

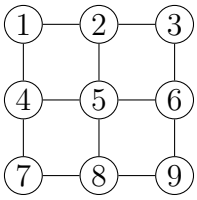

図 2.1: グラフ

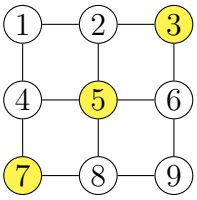

図 2.2: 実行可能解の例

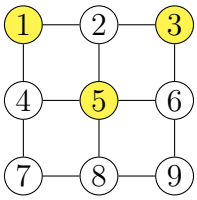

図 2.3: スタート状態

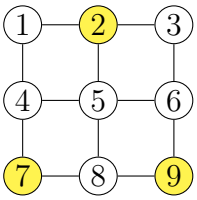

図 2.4: ゴール状態

列 (2.1) の各状態 X_i は基となる独立集合問題の実行可能解を表す．独立集合に含まれるすべての頂点にトークンが配置されていると仮定すると，独立集合遷移問題は，各遷移においてトークンを遷移制約を満たすように移動させる問題と考えることもできる．

独立集合遷移問題の遷移制約は，**トークンジャンピング** (Token Jumping; TJ) と**トークンスライディング** (Token Sliding; TS) の 2 種類が広く研究されている．TJ 型遷移制約は，各遷移においてちょうど 1 個のトークンが他の頂点へ移動することを意味する．一方 TS 型遷移制約は，各遷移においてちょうど 1 個のトークンが隣接する頂点へ移動することを意味する．

例として，図 2.1 に示したグラフが与えられた場合のサイズ 3 の独立集合問題を基とする TJ 型独立集合遷移問題を解く．ここで，スタート状態を図 2.3，ゴール状態を図 2.4 とする．解となる遷移系列を図 2.5 に示す．この遷移系列では，スタート状態からゴール状態まで 3 回の遷移で到達可能である．各遷移においてちょうど 1 個のトークンが他の頂点へ移動しており，TJ 型遷移制約を満たしていることがわかる．また，各状態 X_i が基となる独立集合問題の実行可能解であることがわかる．

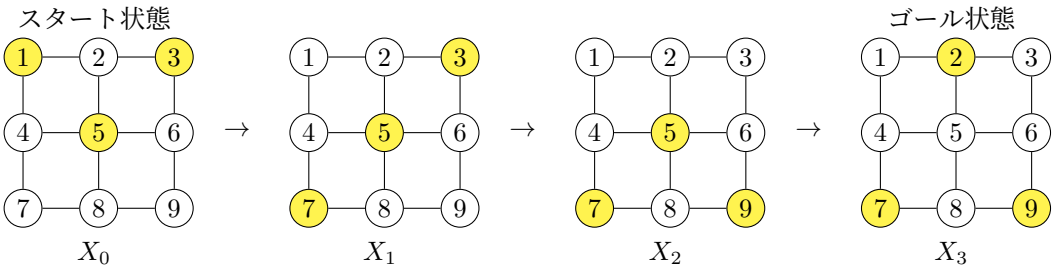

図 2.5: TJ 型独立集合遷移問題の例

第3章　解集合プログラミング

解集合プログラミング (Answer Set Programming; ASP [1, 8, 14, 23]) 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**の集合である $(0 \leq m \leq n)$.

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

各 a_i はアトム，not は**デフォルトの否定**，カンマ (“,”) は連言を表す．“:-” の左側は**ヘッド**，右側は**ボディ**とよばれる．ピリオド (“.”) はルールの終わりを表す．ルール (3.1) 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 が成り立つことである．ボディが空のルールは**ファクト**とよばれ，“:-” を省略して a_0. と記述できる．ヘッドが空のルールは**一貫性制約**とよばれる．

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

空のヘッドは矛盾を表す．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と矛盾する，すなわち，a_1 と a_2 が両方同時に成り立つことはないことを意味する．

組合せ問題を記述する際に便利な拡張構文として，**選択子**と**個数制約**がある．選択子 “{a_1; ...;a_n}.” をファクトとして記述すると，アトムの集合 $\{a_1, \ldots, a_n\}$ の任意の部分集合が成り立つことを意味する．また，選択子の両端に選択可能な個数の上下限を記述することで，個数制約を記述できる．

例えば，“:- not lb {a_1; ...;a_n} ub.” と記述すると，$a_1, \ldots, a_n$ のうち，lb 個以上 ub 個以下が成り立つことを意味する．選択子と個数制約の中括弧内には，アトムの他にリテラル ℓ や**条件付きリテラル** ℓ_0:ℓ_1, ...,ℓ_n も記述可能である．リテラル ℓ とは，アトム a またはその否定 not a である．条件付きリテラル ℓ_0:ℓ_1, ...,ℓ_n は，$\ell_1, \ldots, \ell_n$ が成り立つとき，ℓ_0 が成り立つことを表す．

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 [8] に基づく解集合を計算するシステムである．近年，*clingo* [1]，*DLV* [2]，*WASP* [3] 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．

組合せ遷移問題への ASP システムの応用の一例として，汎用組合せ遷移ソルバー *recongo* が挙げられる．*recongo* は，組合せ遷移問題のインスタンスと，それを解くための ASP 符号化を入力として，*clingo* を用いて組合せ遷移問題の解を求める，ASP ベースの組合せ遷移ソルバーである．*recongo* は，組合せ遷移問題に対して，制限された長さ ℓ の遷移系列を求める問題を論理プログラム φ_ℓ として表現し，それを ASP システムを使って ℓ の値をインクリメンタルに変化させながら繰り返し解くことにより到達性の検査を行う．

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

第4章　独立集合遷移問題を解く ASP符号化

図 4.1: 独立集合遷移問題の解法の流れ

本節では，ASP に基づく独立集合遷移問題の解法について述べる．本研究の解法では，サイズ k の独立集合問題を基とする独立集合遷移問題について，ℓ 回の遷移でスタート状態からゴール状態へ，実行可能解のみを経由して到達可能かという部分問題を，$\ell = 0$ から ℓ の値を 1 ずつ増やしながら繰り返し解いていく．ただし，各遷移において TJ 型遷移制約を満たす．ℓ の上限値を L_{MAX} とし，その値は，基となる独立集合問題の全解数から 1 を引いた値と等しい．任意の $\ell(0 \leq \ell \leq L_{MAX})$ に対する部分問題で到達可能でなければ，本来の独立集合遷移問題が到達不能であることを意味する．一方，ある $\ell(0 \leq \ell \leq L_{MAX})$ に対する部分問題で到達可能であれば，本来の独立集合遷移問題においても到達可能であることを意味する．

図 4.1 に独立集合遷移問題の解法の流れを示す．本研究では，ASP システム *clingo* を用いた汎用組合せ遷移ソルバー *recongo* を用いて独立集合遷移問題の解を求める．はじめに，与えられた独立集合遷移問題のインスタンスを変換器を用いて ASP ファクトに変換する．次に，独立集合遷移問題を解く ASP 符号化と，ASP ファクトに変換した独立集合遷移問題のインスタンスを *recongo* へ与える．その後， *recongo* 内で部分問題を生成し，高速 ASP システムで解く処理を繰り返し実行することで解が計算される．

```
1  n(9).
2  e(12).
3
4  node(1). node(2). node(3). node(4). node(5).
5  node(6). node(7). node(8). node(9).
6
7  edge(1,2). edge(1,4). edge(2,3). edge(2,5).
8  edge(3,6). edge(4,5). edge(4,7). edge(5,6).
9  edge(5,8). edge(6,9). edge(7,8). edge(8,9).
10
11 k(3).
12
13 start(1). start(3). start(5).
14
15 goal(2). goal(7). goal(9).
```

コード 4.1: 図 2.1，図 2.3，図 2.4 のファクト表現

4.1　ASP ファクト形式

recongo の入力となるグラフ，スタート状態，ゴール状態，独立集合のサイズは全て ASP ファクト形式で与えられる．図 2.1 のグラフと，図 2.3, 2.4 のスタート状態及びゴール状態を ASP ファクト形式で表したものをコード 4.1 に示す．コード 4.1 において，1–2 行目では，アトム `n/1`，アトム `e/1` でグラフの頂点数及び辺の数を表す．4–5 行目では，アトム `node/1` で頂点番号 1 から 9 までの 9 個の頂点を表す．7–9 行目では，アトム `edge/2` で 12 本の辺を表しており，`edge(X,Y)` は，頂点 `X` と頂点 `Y` の間に辺が存在することを表す．11 行目ではアトム `k/1` で独立集合のサイズを表す．13 行目では，アトム `start/1` で X_s を表しており，`start(X)` は X_s において頂点 `X` にトークンが配置されていることを表す．15 行目では，アトム `goal/1` で X_g を表しており，`goal(X)` は X_g において頂点 `X` にトークンが配置されていることを表す．

4.2　既存符号化

TJ 型独立集合遷移問題の制約は，スタート制約，ゴール制約，独立集合問題の制約，遷移制約によって表される．スタート制約は，X_0 が X_s と一致することを意味する．ゴール制約は，X_ℓ が X_g と一致することを意味する．独立集合問題の制約は，各状態においてグラフ G がサイズ k の独立集合をもつことを表す．遷移制約は，各遷移において TJ 型遷移制約を満たすことを表す．

```
1  #program base.
2  :- not in(X,0), start(X).
3
4  #program step(t).
5  K { in(X,t): node(X) } K :- k(K).
6  :- in(X,t), in(Y,t), edge(X,Y).
7
8  token_removed(X,t) :- in(X,t-1), not in(X,t), t > 0.
9  :- not 1 { token_removed(X,t) } 1, t > 0.
10
11 #program check(t).
12 :- not in(X,t), goal(X), query(t).
```

コード 4.2: 既存符号化

既存符号化は，各状態においてトークンを配置する頂点に着目しているという特徴がある．既存符号化をコード 4.2 に示す．1 行目の宣言子 `#program` は，1 つのプログラムを複数のサブプログラムに分割するために記述する．コード 4.2 は `base`，`step(t)`，`check(t)` の 3 つのサブプログラムに分割される．`step(t)`，`check(t)` の引数 `t` は X_s から $X_{\mathtt{t}}$ までの遷移長を表す定数パラメータである．

サブプログラム `base` では，X_0 において満たすべき制約を記述する．2 行目のルールは，アトム `in/2` を用いてスタート制約を表す．`in(X,t)` は，$X_{\mathtt{t}}$ においてトークンが頂点 `X` に置かれていることを表す．

サブプログラム `step(t)` では，各状態 $X_{\mathtt{t}}$ において満たすべき制約を記述する．5–6 行目は，独立集合問題の制約を表している．5 行目のルールは，頂点を `K` 個選択してトークンを置くことを表す．6 行目のルールは，互いに隣接する頂点 `X` と頂点 `Y` には同時にトークンが配置されることはないことを表す．8–9 行目は，遷移制約を表している．8 行目では，アトム `token_removed/2` を定義している．`token_removed(X,t)` は，$X_{\mathtt{t}-1}$ において頂点 `X` に配置されているトークンが移動したことを表す．9 行目のルールは，$\mathtt{t} > 0$ のとき，移動するトークンはちょうど 1 つであることを表す．

サブプログラム `check(t)` では，プログラムの終了条件を記述する．12 行目のルールは，アトム `query/1` を用いてゴール制約を表している．$X_{\mathtt{t}-1} \to X_t$ において，`query(t-1)` に 0 が割り当てられる．そのため，$X_{\mathtt{t}-1}$ においてゴール制約が成り立つことはない．

```
1  #program base.
2  in(X,0) :- start(X).
3
4  #program step(t).
5  1 { token_removed(X,t) : in(X,t-1) } 1 :- t > 0.
6  in(X,t) :- in(X,t-1), not token_removed(X,t), t > 0.
7  1 { in(X,t) : node(X), not in(X,t-1) } 1 :- t > 0.
8
9  :- in(X,t), in(Y,t), edge(X,Y).
10
11 #program check(t).
12 :- not in(X,t), goal(X), query(t).
```

コード 4.3: 提案符号化

4.3　提案符号化

TJ 型独立集合遷移問題は，各遷移において移動するトークンは必ず 1 つであるため，それ以外のトークンの配置について考慮する必要はない．そこで提案符号化は，各遷移において移動させるトークンに着目しているという特徴がある．

提案符号化をコード 4.3 に示す．コード 4.3 はコード 4.2 と同様に，`base`，`step(t)`，`check(t)` の 3 つのサブプログラムに分割される．2 行目では，$\mathtt{t} = 0$ におけるアトム `in/2` を定義することで，スタート制約を表す．5–7 行目は，遷移制約を表す．5 行目のルールは，$\mathtt{t} > 0$ のとき，$X_{\mathtt{t}-1}$ で配置されていたトークンから移動させるものを 1 つ選択することを表す．6 行目のルールは，$\mathtt{t} > 0$ のとき，選択されなかったトークンは移動しないことを表す．7 行目のルールは，$\mathtt{t} > 0$ のとき，$X_{\mathtt{t}-1}$ でトークンが置かれていない頂点から 1 つ選択し，そこにトークンを配置することを表す．9 行目は，独立集合問題の制約を表している．提案符号化は，各遷移において 1 つのトークンを移動させ，それ以外のすべてのトークンは移動させないため，各状態においてトークンの数が変化することはない．そのため，独立集合のサイズについての制約は必要なく，9 行目のルールだけで独立集合問題の制約が表せる．

第5章　実行実験

本節では既存符号化，提案符号化に対し，以下の既存のヒント制約を適用する全ての組合せ 16 通りに対する性能評価をするための実行実験を行った．

- X_s からの距離に関する制約 (d1):
  X_s ではトークンが配置されており，かつ X_t では配置されていない頂点の数は高々t 個であることを表す制約である．
- X_g からの距離に関する制約 (d2):
  X_t において，それより前の各状態 $X_T(T \in \{0, \ldots, t-1\})$ に対して，X_g ではトークンが配置されており，かつ X_T では配置されていない頂点の数は高々$t - T$ 個であることを表す制約である．
- 後戻り禁止の制約 (t1):
  X_{t-1} でトークンが頂点 X から頂点 Y へ移動したとき，次の状態 X_t で Y から X にトークンが戻ることを禁止する制約である．
- トークンの移動に関する制約 (t2):
  $X_{t-2} \to X_{t-1}$ で移動したトークンが，次の遷移 $X_{t-1} \to X_t$ で移動することを禁止する制約である．

本実験で使用するベンチマーク問題は，国際組合せ遷移競技会 CoRe Challenge 2022 で使用されたベンチマーク問題集 [1]の全 369 問である．本実験で使用したソルバーは *recongo* のバージョン 0.3.0 であり，使用した ASP システムは *clingo* のバージョン 5.7.1 である．1 問あたりの制限時間は 10 分とした．実行環境は Mac mini，Apple M2，24GB である．実行実験を行う際には，基となる独立集合問題の全解数が判明している場合，その値を L_{MAX} の値に設定する．全解数が判明していない問題では，L_{MAX}

[1] https://github.com/core-challenge/2022benchmark

の値は設定しない．今回使用するベンチマーク問題のうち，全解数が判明している問題は全 369 問中 23 問である．

表 5.1，5.2 に，既存符号化および提案符号化について前述の 16 通りの実行実験で解けた問題数を示す．左から，問題ファミリー，各ヒント制約の組合せごとに解けた問題数を表す．表 5.1，5.2 中の noHint はヒント制約を使用しなかった場合の実行実験を表す．ヒント制約を使用しなかった場合，既存符号化と提案はそれぞれ 194 問，214 問解いた．既存符号化において，全てのヒントを適用した場合に 234 問解き，最も多くの問題を解いた．提案符号化において，`d1`，`t1` を適用させた場合と，`d1`，`t1`，`t2` を適用させた場合に 240 問解き，最も多くの問題を解いた．

既存符号化のカクタスプロット，提案符号化のカクタスプロットをそれぞれ図 5.1，図 5.2 に示す．それぞれの符号化について，ヒントを適用させない場合，及び最も多くの問題数を最も速く解いたヒントの組合せを適用させた場合のカクタスプロットを図 5.3 に示す．横軸は問題を解くのに要した CPU 時間，縦軸は解けた問題数を表す．グラフが上によるほど多くの問題を解けたことを意味し，左によるほどより速く解けたことを意味する．図 5.3 より，`d1`，`t1`，`t2` を適用した提案符号化が最も良い性能を示した．

表 5.3 に，ヒント制約を使用しない場合において提案符号化だけが解けた問題の詳細を示す．左から，提案符号化だけが解けた問題，その頂点数，辺数，独立集合のサイズ，遷移長を表す．表中の ‘-’ は，その問題が到達不能であることを表す．表中の赤字で示された数値は，その数値が全 369 問の中央値より大きいことを表す．提案符号化が解けた問題のグラフの頂点数は 6–40,000 個，辺数は 7–1,646,700 本である．また，平均値はそれぞれ 582.45，43,612.50 であり，中央値はそれぞれ 202，3,247 である．表 5.3 より，提案符号化だけが解いた 20 問の問題のうち 17 問の頂点数と辺数が，ベンチマーク問題全体の頂点数と辺数の中央値よりも大きいことが確認でき，グラフの規模が大きいほど 2 つの符号化の性能の差が出やすいことがわかる．これは，既存符号化が各状態において，トークンを配置する頂点に着目しているのに対し，提案符号化は移動させるトークンに着目しているからだと考えられる．さらに，提案符号化だけが解けた問題のうち半分が問題ファミリー queen の問題であることが確認できる．queen の問題は頂点数に対して独立集合のサイズが小さいという特徴がある．これは，既存符号化は頂点数が多いほど可能なトークンの配置が増え，提案符号化は独立集合のサイズが小さいほど移動させる

表 5.1: 解けた問題数 (既存符号化)

問題ファミリー		noHint	d1	d2	t1	t2	d1d2	d1t1	d1t2	d2t1	d2t2	t1t2	d1d2t1	d1d2t2	d1t1t2	d2t1t2	d1d2t1t2
color04	(202)	170	188	193	168	167	194	189	191	190	189	168	195	196	187	188	196
grid	(49)	5	6	5	5	5	7	7	5	7	5	8	5	6	12	8	6
handcrafted	(6)	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
power	(17)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
queen	(48)	10	19	22	10	10	23	19	19	22	22	10	23	22	18	22	23
sp	(30)	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
square	(17)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
合計	(369)	194	222	229	192	191	233	224	224	228	225	195	232	233	226	227	**234**

表 5.2: 解けた問題数 (提案符号化)

問題ファミリー		noHint	d1	d2	t1	t2	d1d2	d1t1	d1t2	d2t1	d2t2	t1t2	d1d2t1	d1d2t2	d1t1t2	d2t1t2	d1d2t1t2
color04	(202)	177	199	198	177	177	196	199	199	197	198	177	196	197	199	200	196
grid	(49)	7	7	5	7	5	5	7	6	7	5	11	6	5	8	7	6
handcrafted	(6)	6	6	6	6	6	6	6	6	6	5	6	6	5	6	5	5
power	(17)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
queen	(48)	20	23	22	20	19	23	24	23	23	22	20	24	23	24	23	25
sp	(30)	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
square	(17)	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
合計	(369)	214	239	234	213	210	233	**240**	237	236	233	217	235	233	**240**	238	235

トークンの選択肢が減ることで，その性能の差がより顕著に現れたためであると考えられる．

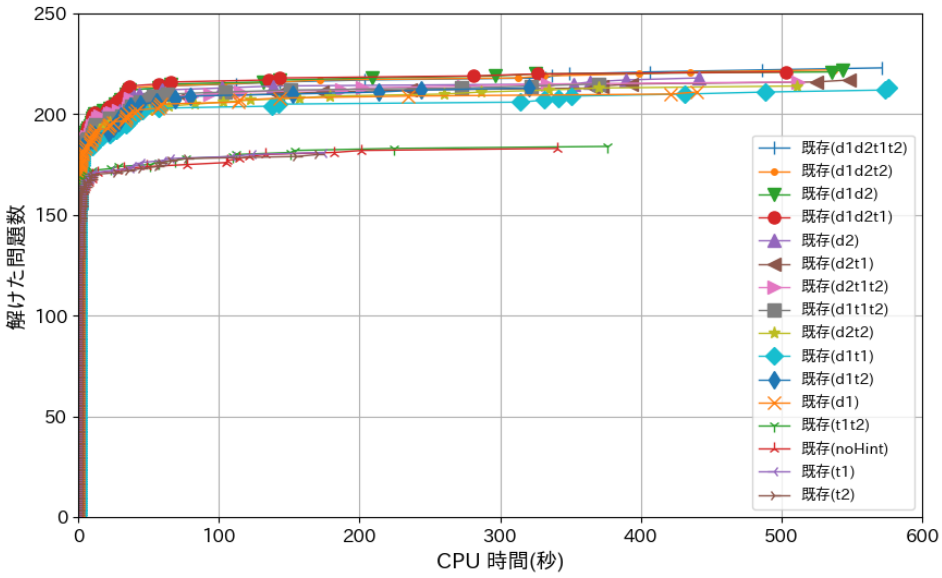

図 5.1: 既存符号化のカクタスプロット

表 5.3: 提案符号化だけが解いた問題の詳細（noHint）

問題	頂点数	辺数	サイズ	遷移長	CPU 時間
grid010x010_02_0495	100	180	49	-	71.838
grid010x010_03_6844	100	180	49	-	77.117
queen030x030_01_3761	**900**	**43,210**	29	15	75.600
queen040x040_01_6291	**1,600**	**103,480**	39	13	173.070
queen040x040_02_9487	**1,600**	**103,480**	39	14	111.055
queen040x040_03_7037	**1,600**	**103,480**	39	14	366.886
queen050x050_01_6006	**2,500**	**203,350**	49	13	145.564
queen050x050_04_9047	**2,500**	**203,350**	49	12	73.147
queen060x060_02_6710	**3,600**	**352,820**	59	9	48.292
queen060x060_03_1394	**3,600**	**352,820**	59	9	69.007
queen060x060_04_3799	**3,600**	**352,820**	59	9	43.340
queen100x100_02_1699	**10,000**	**1,646,700**	99	6	156.186
5-FullIns_4_02	**1,085**	**11,395**	534	6	1.402
DSJR500.1_02	**500**	**3,555**	81	13	8.143
hc-square-004-002_01	44	63	18	90	146.475
le450_15d_02	**450**	**16,750**	40	6	0.246
le450_25a_02	**450**	**8,260**	90	12	2.410
school1_02	**385**	**19,095**	40	11	1.055
wap03a_01	**4,730**	**286,722**	296	10	56.848
wap08a_01	**1,870**	**104,176**	150	13	27.813

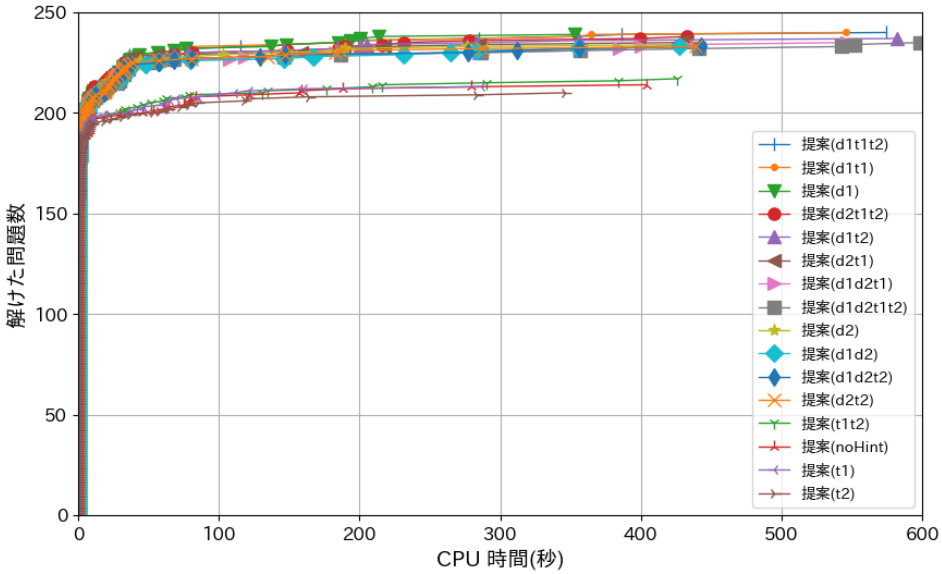

図 5.2: 提案符号化のカクタスプロット

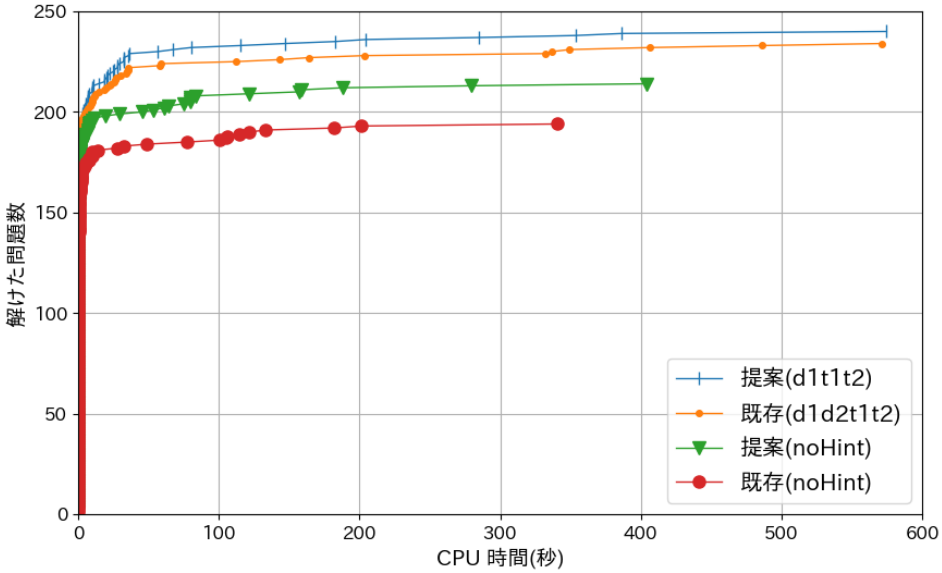

図 5.3: 既存，提案符号化のカクタスプロット

第6章　関連研究

本研究で使用した *recongo* は，ASP に基づく組合せ遷移ソルバーであり，CoRe Challenge 2023 の複数部門で優勝するなど優れた性能を示している．CoRe Challenge 2023 において，他にも組合せ遷移問題を解く汎用ソルバーがいくつか開発された．中でも，プランニングに基づくソルバー *paris* [6] や，ゼロサプレス型二分決定グラフ（ZDD）に基づくソルバー *ddreconf* [17] は優れた性能を示している．CoRe Challenge 2023 においてどのソルバーも解けなかった問題で，提案符号化と *recongo* が解いた問題がある一方，*paris* と *ddreconf* が解いた問題で，本研究の解法で解けなかった問題もある．それらの問題は比較的遷移長が長いという傾向があり，*paris* や *ddreconf* の解法を参考に提案符号化のさらなる改良をすることは，今後の研究課題の一つである．特に，提案符号化のサブプログラム `step(t)` における独立集合問題の制約を無くし，プランニングの手法により近づけることで *paris* のように遷移長の長い問題を効率的に解けるようになると予想される．各状態 X_t において，X_{t-1} で配置されているトークンのうち 1 個を取り除き，トークンの配置されている頂点とその隣接頂点以外のどこかの頂点にトークンを配置することで，アクションによって状態を遷移させるプランニングの特徴を取り入れることができる．

ASP に基づく組合せ遷移問題の解法についても研究が進んでいる．ASP に基づく TJ 型独立集合遷移問題の ASP 解法については，文献 [28, 29] で研究されている．この研究では，本研究で用いた *recongo* と既存符号化を提案しており，組合せ遷移問題に対して初めて ASP を用いている．ASP に基づく支配集合遷移問題の解法については，文献 [21] で研究されている．この研究では，遷移制約として TJ や TS の他に，各遷移においてちょうど 1 個のトークンが追加あるいは削除される TAR（Token Additional/Removal）を用いた到達可能性判定問題の ASP 符号化を提案している．本研究で用いた TJ 以外の遷移制約の実装も，今後の重要な研究課題の一つである．

遷移制約 k-Jump [12] は，各遷移においてちょうど 1 個のトークンが距離 k 以内にある他の頂点へ移動することを意味する．TJ，TS は $D(G)$ をグラフ G の直径とすると，それぞれ $D(G)$-Jump，1-Jump に相当する．そのため，TJ 型遷移制約を考慮した提案符号化のシンプルな拡張によって TS 型，k-Jump 型の独立集合遷移問題を解く符号化を実装できると予想される．

第7章　結論

本研究では，独立集合遷移問題を解く既存の ASP 符号化を改良した．また，既存符号化と提案符号化に対して，4 種類のヒント制約の全ての組合わせ 16 通りを適用した場合の性能評価実験を行った．実行実験の結果，提案符号化が既存符号化に比べ多くの問題を高速に解くことができ，その優位性を確認できた．また，提案符号化がヒント制約 `d1`，`t1` を適用した場合と `d1`，`t1`，`t2` を適用した場合に 369 問中 240 問を解き，その優位性を確認できた．

今後の課題として，提案符号化に対して，効果があるヒント制約の特徴に関する調査が挙げられる．また，ASP 符号化のさらなる改良，ヒューリスティックを考慮した既存のヒント制約 h1 を用いた場合の性能比較，及び頂点間の距離を考慮した遷移制約，k-Jump を用いた独立集合遷移問題を解く ASP 符号化の開発は，今後の重要な研究課題である．

謝辞

本研究にあたり，熱心なご指導を頂いた名古屋大学の番原 睦則教授に深く感謝申し上げます．そして様々なご意見をいただきました番原研究室の皆様に感謝申し上げます．最後に，これまで支えていただいた家族と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] Marthe Bonamy, Paul Dorbec, and Paul Ouvrard. Dominating sets reconfiguration under token sliding. *Discrete Applied Mathematics*, Vol. 301, pp. 6–18, 2021.

[3] Paul S. Bonsma and Luis Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, Vol. 410, No. 50, pp. 5215–5226, 2009.

[4] Richard C. Brewster, Sean McGuinness, Benjamin R. Moore, and Jonathan A. Noel. A dichotomy theorem for circular colouring reconfiguration. *Theoretical Computer Science*, Vol. 639, pp. 1–13, 2016.

[5] Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, Vol. 67, No. 1, pp. 69–82, 2011.

[6] Remo Christen, Salomé Eriksson, Michael Katz, Christian Muise, Alice Petrov, Florian Pommerening, Jendrik Seipp, Silvan Sievers, and David Speck. PARIS: planning algorithms for reconfiguring independent sets. In Kobi Gal, Ann Nowé, Grzegorz J. Nalepa, Roy Fairstein, and Roxana Radulescu, editors, *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, Vol. 372 of *Frontiers in Artificial Intelligence and Applications*, pp. 453–460. IOS Press, 2023.

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

[8] 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.

[9] Parikshit Gopalan, Phokion G. Kolaitis, Elitza N. Maneva, and Christos H. Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM Journal on Computing*, Vol. 38, No. 6, pp. 2330–2355, 2009.

[10] Ruth Haas and Karen Seyffarth. The k-dominating graph. *Graphs and Combinatorics*, Vol. 30, No. 3, pp. 609–617, 2014.

[11] Arash Haddadan, Takehiro Ito, Amer E. Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *Theoretical Computer Science*, Vol. 651, pp. 37–49, 2016.

[12] Hiroki Hatano, Naoki Kitamura, Taisuke Izumi, Takehiro Ito, and Toshimitsu Masuzawa. Independent set reconfiguration under bounded-hop token. *Computing Research Repository*, Vol. abs/2407.11768, , 2024.

[13] Takahiro Hirate, Mutsunori Banbara, Katsumi Inoue, Xiao-Nan Lu, Hidetomo Nabeshima, Torsten Schaub, Takehide Soh, and Naoyuki Tamura. Hamiltonian cycle reconfiguration with answer set programming. In Sarah Alice Gaggl, Maria Vanina Martinez, and Magdalena Ortiz, editors, *Proceedings of the 18th Edition of the European Conference on Logics in Artificial Intelligence (JELIA 2023)*, Vol. 14281 of *LNCS*, pp. 262–277. Springer, 2023.

[14] 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.

[15] Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, Vol. 412, No. 12-14, pp. 1054–1065, 2011.

[16] Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, Vol. 412, No. 12-14, pp. 1054–1065, 2011.

[17] Takehiro Ito, Jun Kawahara, Yu Nakahata, Takehide Soh, Akira Suzuki, Junichi Teruyama, and Takahisa Toda. Zdd-based algorithmic framework for solving shortest reconfiguration problems. In André A. Ciré, editor, *Proceedings of the 20th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2023)*, Vol. 13884 of *LNCS*, pp. 167–183. Springer, 2023.

[18] Takehiro Ito, Hirotaka Ono, and Yota Otachi. Reconfiguration of cliques in a graph. In Rahul Jain, Sanjay Jain, and Frank Stephan, editors, *Proceedings of the 12th Annual Conference on Theory and Applications of Models of Computation (TAMC 2015)*, Vol. 9076 of *LNCS*, pp. 212–223. Springer, 2015.

[19] Marcin Kaminski, Paul Medvedev, and Martin Milanic. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, Vol. 439, pp. 9–15, 2012.

[20] Masato Kato, Mutsunori Banbara, Torsten Schaub, Takehide Soh, and Naoyuki Tamura. Dominating set reconfiguration with answer set programming. *Theory and Practice of Logic Programming*, Vol. 24, No. 4, p. 755–771, 2024.

[21] Masato Kato, Torsten Schaub, Takehide Soh, Naoyuki Tamura, and Mutsunori Banbara. Dominating set reconfiguration with answer set programming. In *ICLP 2024, to appear.*

[22] Amer E. Mouawad, Naomi Nishimura, Vinayak Pathak, and Venkatesh Raman. Shortest reconfiguration paths in the solution space of boolean formulas. *SIAM Journal on Discrete Mathematics*, Vol. 31, No. 3, pp. 2185–2200, 2017.

[23] 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.

[24] Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, Vol. 11, No. 4, p. 52, 2018.

[25] Akira Suzuki, Amer E. Mouawad, and Naomi Nishimura. Reconfiguration of dominating sets. *Journal of Combinatorial Optimization*, Vol. 32, No. 4, pp. 1182–1195, 2016.

[26] Asahi Takaoka. Complexity of hamiltonian cycle reconfiguration. *Algorithms*, Vol. 11, No. 9, p. 140, 2018.

[27] Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, Vol. 409 of *London Mathematical Society Lecture Note Series*, pp. 127–160. Cambridge University Press, 2013.

[28] Yuya Yamada, Mutsunori Banbara, Katsumi Inoue, and Torsten Schaub. Recongo: Bounded combinatorial reconfiguration with answer set programming. In Sarah Alice Gaggl, Maria Vanina Martinez, and Magdalena Ortiz, editors, *Proceedings of the 18th Edition of the European Conference on Logics in Artificial Intelligence (JELIA 2023)*, Vol. 14281 of *LNCS*, pp. 278–286. Springer, 2023.

[29] Yuya Yamada, Mutsunori Banbara, Katsumi Inoue, Torsten Schaub, and Ryuhei Uehara. Combinatorial reconfiguration with answer set programming: Algorithms, encodings, and empirical analysis. In Ryuhei Uehara, Katsuhisa Yamanaka, and Hsu-Chun Yen, editors, *Proceedings of the 18th International Conference and Workshops on Algorithms and Computation (WALCOM 2024)*, Vol. 14549 of *Lecture Notes in Computer Science*, pp. 242–256. Springer, 2024.