解集合プログラミングを用いた遷移コストと遷移長を考慮したパレート最適な組合せ遷移

高田 和紀　楊 東　番原 睦則

本稿では，遷移コストと遷移長の両方を考慮した多目的組合せ遷移最適化問題 (MO-CROP) を新たに提案し，解集合プログラミング (ASP) を用いた解法について述べる．MO-CROP とは，基となる組合せ問題，その 2 つの実行可能解，複数の重み付き遷移制約が与えられたとき，一方から他方へ，遷移コストと遷移長の両方を考慮したパレート最適な遷移系列を求める問題である．従来の組合せ遷移問題が，ただ一つの遷移制約を考慮するのに対し，MO-CROP では複数の重み付き遷移制約を考慮する点が大きく異なる．提案解法は，遷移長を制限した単目的組合せ遷移最適化問題を，遷移長を増やしながら繰り返し解くことによって最適遷移系列を求める．提案アルゴリズムの特長は，遷移長の上限値を自動で適切に更新することによって，パレート最適な遷移系列の全列挙を保証できる点である．提案手法の有効性を評価するために，独立集合遷移に基づく MO-CROP を解く ASP 符号化を新たに考案し，CoRe Challenge 2022 のベンチマーク問題集を用いた実験結果について述べる．

1　はじめに

組合せ遷移 (Combinatorial Reconfiguration; [17] [25] [31]) とは，組合せ問題の実行可能解が形成する解空間を対象とする研究分野である．解空間はグラフとして定義され，頂点は実行可能解を表し，辺は遷移制約 (隣接関係ともよばれる) を満たす 2 つの実行可能解の組を表す．基となる組合せ問題が実行可能解の存在性を判定するのに対し，組合せ遷移問題は解空間グラフの構造や性質を分析することを主眼とする．

組合せ遷移の到達可能性問題 [17] 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どうかを判定する問題である．本稿ではこの問題を組合せ遷移問題 (Combinatorial Reconfiguration Problem) とよぶ．

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に急速に発展し理論的な基盤が整備されている．特に計算複雑性理論の研究では，基となる組合せ問題が NP 完全の場合，その組合せ遷移問題の多くが PSPACE 完全になることが明らかになっている [17]．そのような例としては，命題論理の充足可能性判定 (SAT) 遷移 [11] [23]，独立集合遷移 [17] [20]，支配集合遷移 [2] [12] [13] [28], グラフ彩色遷移 [3] [4] [5]，クリーク遷移 [19]，ハミルトン閉路遷移 [29]，集合被覆遷移 [17] などがあげられる．

ごく最近では，国際組合せ遷移競技会 CoRe Challenge 2022–2023 [26] が 2 年連続で開催され，組合せ遷移問題の解法アルゴリズムなど実践的な研究も盛んになっている [6] [18] [30] [32] [33]．しかしながら，組合せ遷移の従来研究の大多数は，ただ一つの遷移制約のみを考慮した遷移系列の有無を対象としており，複数の重み付き遷移制約を考慮した研究はほとんどなされていない．

本稿では，遷移コストと遷移長の両方を考慮した**多目的組合せ遷移最適化問題** (Multi-Objective Combinatorial Reconfiguration Optimization Problem; MO-CROP) を新たに提案し，解集合プログラミングを用いた解法アプローチについて述べる．提案する

Pareto-optimal Combinatorial Reconfiguration considering Cost and Length based on Answer Set Programming

Kazuki Takada, Dong Yang, Mutsunori Banbara, 名古屋大学大学院情報学研究科, Graduate School of Informatics, Nagoya University.

MO-CROP は，基となる組合せ問題，その 2 つの実行可能解，複数の重み付き遷移制約が与えられたとき，一方から他方へ，遷移コストと遷移長の両方を考慮したパレート最適な遷移系列を求める問題である．従来の組合せ遷移問題がただ一つの遷移制約を考慮するのに対し，MO-CROP では複数の重み付き遷移制約を考慮する点が大きく異なる．**解集合プログラミング** (Answer Set Programming; ASP [1][10][24]) 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．MO-CROP に対して ASP を用いる利点としては，ASP 言語の高い表現力を生かして基の組合せ問題を簡潔に記述できる点，遷移最適化問題への拡張も容易である点，マルチショット ASP 解法 [9] を利用することで，効率的な遷移系列探索の実装が可能である点があげられる．

本稿の主な貢献および結果は以下の通りである．

1. 複数の重み付き遷移制約をもつ多目的組合せ遷移最適化問題 (MO-CROP) を提案する．この問題は，ただ一つの遷移制約を考慮する従来の組合せ遷移問題の自然な拡張になっており，遷移コストと遷移長のトレードオフを考慮した多様な最適遷移系列を求める問題である．
2. MO-CROP に対する解法アルゴリズムを提案する．提案解法は，遷移長を制限した単目的 CROP を，遷移長を増やしながら繰り返し解くことによって最適遷移系列を求める．提案アルゴリズムの特長は，遷移長の上限値を自動で適切に更新することによって，パレート最適な遷移系列の全列挙を保証できる点である．
3. ASP に基づく MO-CROP の汎用ソルバー *recongo_opt* の設計と実装を詳述する．*recongo_opt* は，ASP システム *clingo* のマルチショット ASP 解法 [9] を用いて解法アルゴリズムを実装することにより，基礎化にかかるオーバーヘッドを抑えつつ，獲得した学習節を再利用した効率的な遷移系列の探索が可能である．
4. MO-CROP のケーススタディとして，トークンジャンピング (TJ) とトークンスライディング (TS) の 2 つの遷移制約をもつ，TJ–TS 型独立集合遷移最適化問題を解く ASP 符号化を示す．この符号化は，従来の TJ 型独立集合遷移問題の ASP 符号化 [33] と，本稿で新しく考案した TS 型独立集合遷移問題を解く ASP 符号化の自然な拡張となっている．
5. 提案アプローチの有効性を評価するために，CoRe Challenge 2022 のベンチマーク問題集 (計 326 問) を用いて実験を行った．その結果，TJ–TS 型独立集合遷移最適化問題 212 問 (全体の 65%) についてパレート最適な遷移系列を全列挙することができた．

提案アプローチは，組合せ遷移の実践的研究，および，ASP の組合せ遷移への応用研究に寄与するものである．本稿の ASP 符号化は ASP システム *clingo* の言語で記述されている．ASP の言語構文については第 3 節で簡単に述べるが，より詳しい説明については文献 [8] を参照いただきたい．

2 準備: 組合せ遷移問題

組合せ遷移問題とは，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，および一つの遷移制約が与えられたとき，遷移系列

$$X_s = X_0 \to X_1 \to X_2 \to \cdots \to X_\ell = X_g \quad (1)$$

が存在するかどうかを判定する問題である．X_s はスタート状態，X_g はゴール状態を表す．各状態 X_i は基となる組合せ問題の実行可能解を表す．$X \to X'$ は遷移制約に従って状態 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．(1) のような遷移系列が存在するとき，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．

組合せ遷移問題の例として独立集合遷移問題を考える．無向グラフ $G = (V, E)$ が与えられたとき，サイズ k の**独立集合** (Independent Set) $V' \subseteq V$ とは，$|\, V' \,| = k$ であり，任意の 2 頂点 $v_1, v_2 \in V'$ について $(v_1, v_2) \notin E$ となるような V の部分集合 V' である．**独立集合問題** (Independent Set Problem) は，与えられたグラフ G と自然数 k に対して，G がサイズ k の独立集合をもつかどうかを判定する問題である．**独立集合遷移問題** (Independent Set Reconfiguration Problem) は，独立集合問題を基とする組合せ遷移問

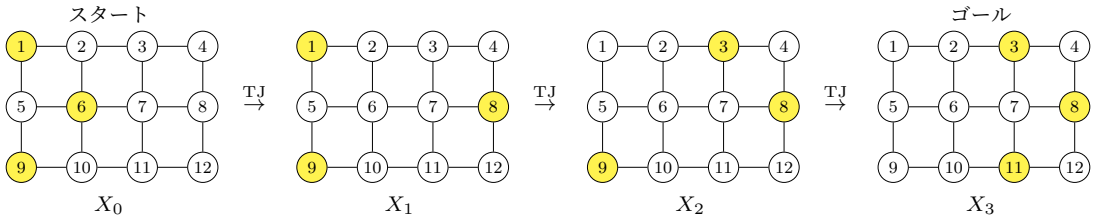

図 1　TJ 型独立集合遷移問題の例

題である．すなわち，遷移系列 (1) の各状態 X_i は独立集合を表す．

独立集合に含まれるすべての頂点にトークン (token) が置かれると仮定すると，独立集合遷移問題は，トークンを移動する問題と考えることもできる．スタート状態の独立集合に含まれるすべての頂点にトークンが置かれている．トークンは遷移制約を満たすように移動させることができる．ただし，トークンが置かれた頂点の集合は，つねに独立集合問題の制約を満たさなければならない．最終的に，トークンの置かれた頂点の集合がゴール状態の独立集合と一致すれば到達可能となる．独立集合遷移問題の遷移制約としては，1 回の遷移でちょうど一つのトークンが移動する**トークンジャンピング** (Token Jumping; TJ) と，1 回の遷移でちょうど一つのトークンが隣接頂点へ移動する**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 型独立集合遷移問題の例を図 1 に示す．独立集合のサイズは $k = 3$ とし，黄色の頂点はトークンが置かれていることを意味する．この例では，3 回の遷移でスタート状態 (X_0) からゴール状態 (X_3) へ到達可能である．各状態 X_i は独立集合問題の制約を満たしている．また各遷移では，ちょうど 1 個のトークンが他の頂点へ移動する遷移制約 TJ を満たしている．

3　準備: 解集合プログラミング

解集合プログラミングは，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 の言語は，一般拡張選言プログラムに基づいている [16]．本節では，そのサブクラスである標準論理プログラム (normal logic program) について説明する．本稿では，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 ($0 \leq m \leq n$)．

$$a_0 \texttt{ :- } a_1\texttt{,}\ldots\texttt{,}a_m\texttt{, not } a_{m+1}\texttt{,}\ldots\texttt{,not } a_n\texttt{.} \quad (2)$$

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．“`:-`” の左側を**ヘッド**，右側を**ボディ**とよぶ．ピリオド (“.”) はルールの終わりを表す．ルール (2) 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は**ファクト**とよばれ，“`:-`” を省略して a_0. と記述される．ヘッドが空のルールは**一貫性制約**とよばれる．

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

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」，すなわち，「a_1 と a_2 が両方同時に成り立つことはない」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 “$\texttt{\{}a_1\texttt{;}\ldots\texttt{;}a_n\texttt{\}}$.” 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．選択子の両端に選択可能な個数の上下限を付けることによって個数制約を記述できる．例えば，“$\texttt{:- not } lb\ \texttt{\{}a_1\texttt{;}\ldots\texttt{;}a_n\texttt{\}}\ ub$.” と書くと，「$a_1, \ldots, a_n$ のうち，lb 個以上 ub 個以下が成り立つ」を意味する．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** $\ell_0\texttt{:}\ell_1\texttt{,}\ldots\texttt{,}\ell_n$ も記述可能である．リテラル ℓ はアトム a またはその否定 $\texttt{not } a$ である．条件付きリテラル $\ell_0\texttt{:}\ell_1\texttt{,}\ldots\texttt{,}\ell_n$ 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．他にも，組合せ最適化問題を解くための最大化関数 `#maximize` や最小化関数

```
k(3).

node(1).  node(2).  node(3). node(4).  node(5).  node(6).
node(7).  node(8).  node(9). node(10). node(11). node(12).

edge(1,2).   edge(1,5).  edge(2,3).  edge(2,6).  edge(3,4).
edge(3,7).   edge(4,8).  edge(5,6).  edge(5,9).  edge(6,7).
edge(6,10).  edge(7,8).  edge(7,11). edge(8,12). edge(9,10).
edge(10,11). edge(11,12).

start(1).  start(6).  start(9).
goal(3).   goal(8).   goal(11).
```

リスト 1　独立集合遷移問題のインスタンス (図 1) の ASP ファクト表現

```
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).
```

リスト 2　TJ 型独立集合遷移問題を解く ASP 符号化 (TJ 符号化)

`#minimize` 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 [10] に基づく解集合を計算するプログラムである．ASP システムの多くは，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**基礎化**したのち，基礎ソルバーを用いて解集合を計算する．本稿で用いる ASP システム *clingo* も，基礎化のためのグラウンダー *gringo* と基礎ソルバー *clasp* をシームレスに結合したものである．

マルチショット ASP 解法は，ファクトやルールを動的に追加・変更・削除しながら一連の論理プログラムを効率良く解くための方法である．*clingo* には，マルチショット ASP 解法のため新しい言語構文として，`#program` 文と `#external` 文が導入されている．前者は `#program` $p(t)$. の形式で，ASP プログラムをいくつかのパラメータ付きサブプログラムに分割するために使用される．p はサブプログラム名を表し，パラメータ t は定数である．この t は省略可能である．`base` はデフォルトのサブプログラムであり，`#program` 文によって明示的に指定されないすべてのファクトとルールを含む．後者は `#external` a. の形式で，外部アトム a の真偽値が (*clingo* の Python API を使って) 後で変更できることを意味する．

論理プログラムの例として，第 2 節の図 1 に示した TJ 型独立集合遷移問題を考える．まず最初に，図 1 の問題インスタンスのファクト形式をリスト 1 に示す．アトム `k/1` は独立集合のサイズを表す．アトム `node/1` は頂点を，`edge/2` は辺をそれぞれ表す．アトム `start/1` はスタート状態に含まれる頂点を，`goal/1` はゴール状態に含まれる頂点をそれぞれ表す．例えば，`start(1)` は頂点 1 がスタート状態の独立集合に含まれることを表している．

つぎに，文献 [33] で提案された TJ 型独立集合遷移問題を解く論理プログラム (ASP 符号化) をリスト 2 に示す．この符号化は，`base`, `step(t)`, `check(t)` の 3 つのサブプログラムから構成されている．`step(t)` と `check(t)` のパラメータ `t` はステップ数 (状態 X_i の i) を表す定数である．サブプログラム `base` はスタート状態に関する制約を表す．アトム `in(X,t)` は，ステップ `t` において頂点 `X` にトークンが置かれていることを意味する．2 行目のルールは，ステップ `0` とスタート状態でトークンの配置が一致することを一貫性制約を用いて強制している．サブプログラム `step(t)` は，独立集合問題の制約と遷移制約を表す．5–6 行目のルールは独立集合問題の制約を表す．5 行目のルールは，各ステップ `t` において，独立集合のサイズがちょうど `K` 個であることを個数制約を用いて強制している．6 行目のルールは，各ステップ `t` において，隣接する頂点 `X` と `Y` が同時に独立集合に含まれることはないことを一貫性制約を用いて強制している．8–9 行目のルールは TJ 遷移制約を表す．8 行目のルールでは，各ステップ `t` で，頂点 `X` からトークンが移動したことを表す補助アトム `token_removed(X,t)` を導入する．9 行目のルールは，各ステップ `t` で，

```
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 :- not 1 { in(Y,t) : edge(X,Y) ; in(Y,t) : edge(Y,X) } 1,
       token_removed(X,t).
11
12 #program check(t).
13 :- not in(X,t), goal(X), query(t).
```

リスト 3　TS 型独立集合遷移問題を解く ASP 符号化 (TS 符号化)

`token_removed(X,t)` が真になる頂点がただ一つであることを個数制約を用いて強制している．サブプログラム `check(t)` はゴール状態に関する制約を表す．12 行目のルールは，ステップ `t` とゴール状態でトークンの配置が一致するかどうかをチェックする．ここで，`query(t)` は`#external` 文によって宣言された外部アトムである．この外部アトムの真理値を *clingo* の Python API を用いて動的に変更することにより，12 行目のルールの活性・非活性を制御できる．

最後に，問題インスタンスのファクト表現 (リスト 1) と独立集合遷移問題を解く ASP 符号化 (リスト 2) を入力として，*clingo* 上に実装された組合せ遷移問題ソルバー *recongo* を実行することで，図 1 の遷移系列を求めることができる．

4　TS 型独立集合遷移問題の ASP 符号化

本節では，TS 型独立集合遷移問題を解く ASP 符号化について述べる．独立集合遷移問題の TS 遷移制約は，TJ 遷移制約の特殊形である．TJ 遷移制約では 1 回の遷移でちょうど一つのトークンが移動するのに対し，TS 遷移制約ではその移動先が (移動元の) 隣接頂点に限定される．TS 型独立集合遷移問題の例を図 2 に示す．この例では，6 回の遷移でスタート状態からゴール状態へ到達可能である．また各状態は独立集合問題の制約を満たしている．また各遷移では，ちょうど 1 個のトークンが隣接頂点へ移動する TS 遷移制約を満たしている．

考案した TS 型独立集合遷移問題を解く ASP 符号化をリスト 3 に示す．TJ 型独立集合遷移問題 (リスト 2) との違いは，10 行目のルールが追加されている点のみである．このルールは，各ステップ `t`，各頂点 `X` について，`X` からトークンが移動するならば，移動先はその隣接頂点であることを個数制約で強制している．これは，互いに隣接する 2 頂点には同時にトークンが置かれないという独立集合の性質を利用しており，既存の TJ 型独立集合遷移問題の ASP 符号化 [33] の自然な拡張となっている．

5　遷移コストと遷移長を考慮した多目的組合せ遷移最適化

多目的組合せ遷移最適化問題 (MO-CROP) 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，複数の重み付き遷移制約が与えられたとき，遷移系列 $X_s = X_0 \to X_1 \to X_2 \to \cdots \to X_\ell = X_g$ のうち，遷移コストと遷移長の両方を考慮したパレート最適な遷移系列を求める問題である．従来の組合せ遷移問題と同様に，X_s はスタート状態，X_g はゴール状態，各状態 X_i は基となる組合せ問題の実行可能解を表す．$X \to X'$ は複数の重み付き遷移制約のいずれか一つに従って状態 X から状態 X' へ 1 回遷移することを表す．**遷移コスト**は各遷移で適用された遷移制約の重みの総和を表す．**遷移長**は遷移系列の長さ ℓ を表す．

提案解法は，与えられた MO-CROP に対して，制限された長さでの最小コスト遷移系列を求める問題を論理式として表現し，その論理式を ASP システム等の汎用ソルバーを使って解くことにより，パレート最適な遷移系列を全列挙する方法である．ソルバーが最適解を求めた場合，制限された長さでの最小コスト遷移系列が得られることを意味する．逆に充足不能と判定した場合，制限された長さでは到達可能な遷移系列が存在しないことを意味する．この場合，遷移系列の長さを増加させた論理式を構成し，再びソルバーによる実行を繰り返す．提案解法は，パレート最適な遷移系列を探すだけで，到達不能の証明はできない不完

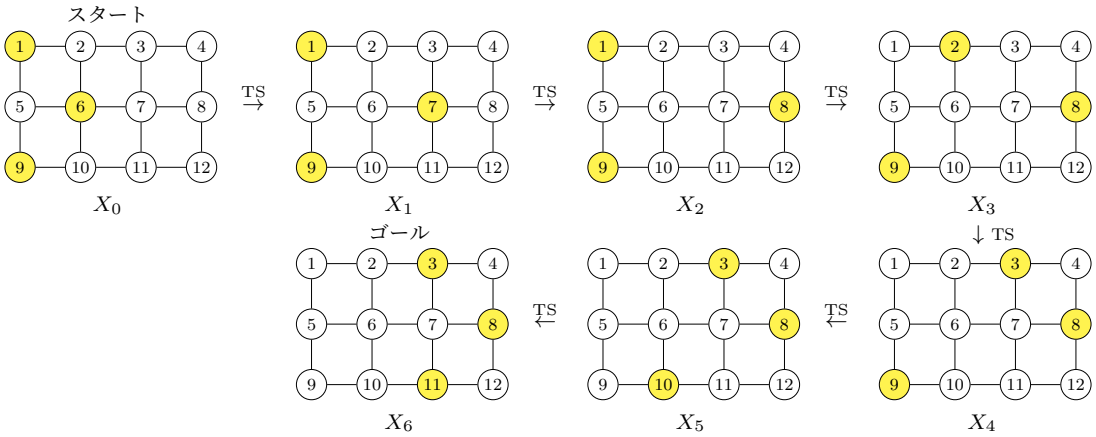

図 2　TS 型独立集合遷移問題の例

全な手続きである．検査すべき遷移系列の長さに 上限が存在する場合 (例えば，基の組合せ問題の直径が既知の場合) には，完全な手続きとなる．

定式化．MO-CROP が与えられたとき，基となる組合せ問題の変数集合を $\boldsymbol{x} = \{x_1, x_2, ..., x_n\}$ で表す．また，各状態 X_t $(t \geq 0)$ の変数集合を $\boldsymbol{x^t} = \{x_1^t, x_2^t, ..., x_n^t\}$ と表記する．スタート状態を表す論理式を $S(\boldsymbol{x})$，基となる組合せ問題の制約を表す論理式を $C(\boldsymbol{x})$，ゴール状態を表す論理式を $G(\boldsymbol{x})$ とする．与えられた m 個の遷移制約を $T_j(\boldsymbol{x}, \boldsymbol{x}')$，その重みを $w_j > 0$ で表す $(1 \leq j \leq m)$．このとき，遷移制約を表す論理式 $T(\boldsymbol{x}, \boldsymbol{x}')$ と遷移コスト $O(\boldsymbol{x}, \boldsymbol{x}')$ は，以下のように表される．

$$T(\boldsymbol{x}, \boldsymbol{x}') = exact\_one\left(T_1(\boldsymbol{x}, \boldsymbol{x}'), \ldots, T_m(\boldsymbol{x}, \boldsymbol{x}')\right)$$

$$O(\boldsymbol{x}, \boldsymbol{x}') = \sum_{j=1}^{m} w_j T_j(\boldsymbol{x}, \boldsymbol{x}')$$

$T(\boldsymbol{x}, \boldsymbol{x}')$ は m 個の遷移制約のうち，ちょうど一つが適用されることを表す．スタート状態から ℓ 回遷移した後の状態 X_ℓ の変数集合 $\boldsymbol{x^\ell}$ は，$S(\boldsymbol{x}^0) \wedge \bigwedge_{t=0}^{\ell} C(\boldsymbol{x}^t) \wedge \bigwedge_{t=1}^{\ell} T(\boldsymbol{x}^{t-1}, \boldsymbol{x}^t)$ を満たす．そして，ゴール状態と一致する最小コスト遷移系列を求めるためには，$G(\boldsymbol{x}^\ell)$ と目的関数 $minimize\left(\sum_{t=1}^{\ell} O(\boldsymbol{x}^{t-1}, \boldsymbol{x}^t)\right)$ を付け加えた論理式 ψ_ℓ を構成すればよい．

$$\psi_\ell = S(\boldsymbol{x}^0) \wedge \bigwedge_{t=0}^{\ell} C(\boldsymbol{x}^t) \wedge \bigwedge_{t=1}^{\ell} T(\boldsymbol{x}^{t-1}, \boldsymbol{x}^t) \wedge G(\boldsymbol{x}^\ell) \wedge minimize\left(\sum_{t=1}^{\ell} O(\boldsymbol{x}^{t-1}, \boldsymbol{x}^t)\right)$$

この ψ_ℓ の最適解が求まると，制限された遷移長 ℓ における最小コスト遷移系列が求まる．遷移コストと遷移長を考慮したパレート最適な遷移系列を列挙するには，遷移長 ℓ を 0 から 1 ずつ増やしながら，ψ_ℓ を繰り返し解き，得られた最小コスト遷移系列のパレート最適性をチェックすればよい．しかしながら，現段階では遷移長の上限がわからないため，パレート最適な遷移系列の全列挙を保証することはできない．つまり，パレートフロントを知ることはできない．この問題を解消するには，得られた遷移コストをもとに，遷移長 ℓ の上限値 I^{max} を適切に更新する必要がある．

パレートフロントの保証．遷移長 ℓ の上限値 I^{max} を更新するための基本的アイデアは，パレート最適な遷移系列が見つかるたびに I^{max} の値を更新し，遷移長 ℓ を 0 から I^{max} まで変化させて探索を行うことで，パレート最適な遷移系列の全列挙を保証する点である．遷移制約の最小の重みを $w_{min} = \min\{w_j : 1 \leq j \leq m\}$ とすると，遷移長 ℓ における遷移系列の遷移コストの下限は ℓw_{min} となる．いま，直近で求まったパレート最適な遷移系列の遷移コストを W とすると，$\ell w_{min} > W$ となるような ℓ では，W を下回る遷移コストは得られず，パレート最適な遷移系列は存在しない．したがって，$\ell w_{min} \leq W$，すなわち $\ell \leq W/w_{min}$ を満たす ℓ まで探索すればよい．以上より，$I^{max} = \lfloor W/w_{min} \rfloor$ と更新することでパレート最適な遷移系列の全列挙を保証できる．

アルゴリズム．提案解法は，MO-CROP が与えら

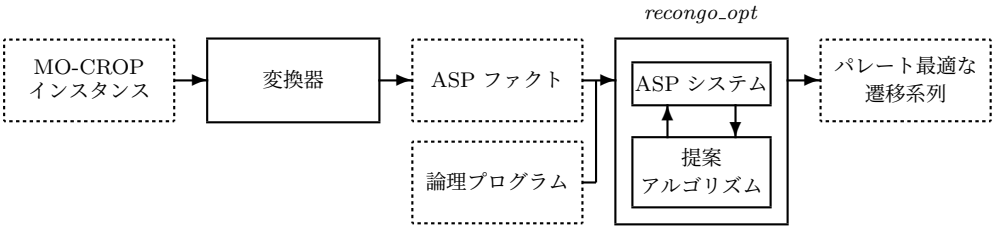

図 3　提案アプローチの構成

れたとき，ψ_ℓ を論理プログラムとして表現し，その論理プログラムを ASP システムで実行することにより，パレート最適な遷移系列を全列挙する．より正確には，ψ_ℓ の ℓ を 0 から 1 ずつ増加させながら繰り返し実行し，ℓ が上限値 I^{max} を超えたとき，パレート最適な遷移系列がすべて得られる．しかしながら，毎回 ψ_ℓ を再構築するのは明らかに非効率である．この問題を解決するために，マルチショット ASP 解法 [9] を用いることで，インクリメンタルに ψ_ℓ を構成する．具体的には，ψ_ℓ は，$\psi_{\ell-1}$ に $C(\boldsymbol{x}^\ell)$，$T(\boldsymbol{x}^{\ell-1}, \boldsymbol{x}^\ell)$，$G(\boldsymbol{x}^\ell)$，$O(\ell) = \mathit{minimize}\ O(\boldsymbol{x}^{\ell-1}, \boldsymbol{x}^\ell)$ を追加し，$G(\boldsymbol{x}^{\ell-1})$ を不活性化することで，インクリメンタルに構成できる．これにより，提案アルゴリズムは，学習節の再利用や基礎化のオーバーヘッド軽減による効率的な最適遷移系列の探索が可能である．なお，ASP システム *clingo* において，$\mathit{minimize} \sum_{t=1}^{\ell} O(\boldsymbol{x}^{t-1}, \boldsymbol{x}^t)$ と $\bigwedge_{t=1}^{\ell} \mathit{minimize}\ O(\boldsymbol{x}^{t-1}, \boldsymbol{x}^t)$ は同値である．

提案アルゴリズムの擬似コードをアルゴリズム 1 に示す．変数 i は各状態 X_i のステップ番号 i を表す．変数 $model^*$ はパレート最適な遷移系列を格納する配列を表し，$cost^*$ は直近に求まったパレート最適な遷移系列の遷移コストを表す．変数 ret, $model$, $cost$ には，ASP システムの実行結果，解集合，その遷移コストが保存される．8 行目で追加される `#external` 文は，ゴール状態を表す $G(\boldsymbol{x}^t)$ の活性・不活性を切り替えるために使われる．各ステップ i において，10–25 行目で ψ_i をインクリメンタルに構成する．その後，26 行目で ASP システムを用いて遷移長 $\ell = i$ における最小コスト遷移系列を求め，27 行目でステップ i の値を 1 増やす．31–38 行目では，得られた遷移系列の遷移コストが直近で求まったパレート最適な遷移系列の遷移コストより小さくなるとき，その遷移系列を $model^*$ に格納し，$cost^*$ と I^{max} の更新を行う．最後に，$i > I^{max}$ となるとき，41 行目で $model^*$ を出力する．

ソルバーの実装．提案アルゴリズムを高速 ASP システム *clingo* に実装したソルバー *recongo_opt* を開発した．*recongo_opt* は，与えられた MO-CROP インスタンス (ASP ファクト形式) と MO-CROP を解く ASP 符号化を入力とし，高速 ASP システム *clingo* の Python API を用いて実装した提案アルゴリズムを用いて解を求める (図 3 参照)．

6　独立集合の多目的遷移最適化

多目的独立集合遷移最適化問題とは，独立集合問題に基づく多目的組合せ遷移最適化問題である．多目的独立集合遷移最適化問題の例を図 4 に示す．遷移制約としては TJ と TS の両方を考慮し，重みをそれぞれ 5 と 2 とする．この例は，全部で 4 つあるパレート最適な遷移系列のうち，(遷移コスト, 遷移長) = (13,5) の最適遷移系列を示している．すなわち，5 回の遷移でスタート状態からゴール状態へ到達し (1 回目が TJ で 2–5 回目が TS)，その遷移コストは 13 である．各状態は独立集合問題の制約を満たしている．また各遷移では，TJ か TS のいずれかに従ってちょうど 1 個のトークンが移動している．他の 3 つのパレート最適な遷移系列は，(遷移コスト, 遷移長) = (12,6), (14,4), (15,3) である．そのうち，(15,3) は TJ のみの遷移系列 (図 1) に対応し，(12,6) は TS のみの遷移系列 (図 2) に対応する．このように，多目的独立集合遷移最適化問題では，従来の独立集合遷移問題と比較して，(13,5) や (14,4) のような遷移コストと遷移長のトレードオフを考慮した多様な最適遷移系列を得ることができる．

多目的独立集合遷移最適化問題を解く TJ–TS 符号化をリスト 4 に示す．従来の独立集合遷移問題を解く

Algorithm 1 提案アルゴリズムの疑似コード

Input P: problem instance of ASP fact format
Input $S(\boldsymbol{x}^0)$, $C(\boldsymbol{x}^t)$, $T(\boldsymbol{x}^{t-1}, \boldsymbol{x}^t)$, $G(\boldsymbol{x}^t)$, $O(t)$: logic program
Parameter I^{max}: the maximum number of steps [*none*]

1: $ctl \leftarrow$ create an object of ASP solver
2: $i \leftarrow 0$
3: $ret \leftarrow none$
4: $model \leftarrow none$
5: $model^* \leftarrow$ an empty list
6: $cost \leftarrow cost^* \leftarrow none$
7: $w_{min} \leftarrow none$
8: add a statement "`#external query(`t`).`" to $G(\boldsymbol{x}^t)$
9: **while** ($I^{max} = none$ **or** $i \leq I^{max}$) **do**
10: $parts \leftarrow$ an empty list
11: $parts.append(C(\boldsymbol{x}^i))$
12: $parts.append(G(\boldsymbol{x}^i))$
13: $parts.append(O(i))$
14: **if** $i > 0$ **then**
15: $parts.append(T(\boldsymbol{x}^{i-1}, \boldsymbol{x}^i))$
16: $ctl.release\_external($`query(`$i-1$`)`$)$ {deactivating $G(\boldsymbol{x}^{i-1})$}
17: **else**
18: $parts.append(P)$
19: $parts.append(S(\boldsymbol{x}^0))$
20: **end if**
21: $ctl.ground(parts)$
22: **if** $i = 0$ **then**
23: $w_{min} \leftarrow get\_minimum\_weight(O(i))$
24: **end if**
25: $ctl.assign\_external($`query(`i`)`$,$ **true**$)$ {activating $G(\boldsymbol{x}^i)$}
26: $(ret,\ model) = ctl.solve()$
27: $i \leftarrow i + 1$
28: **if** $ret \neq$ SAT **then**
29: *continue*
30: **end if**
31: $cost \leftarrow model.cost[0]$
32: **if** $cost^* = none$ **or** $cost < cost^*$ **then**
33: $cost^* \leftarrow cost$
34: $model^*.append(model)$
35: **if** $I^{max} = none$ **or** $I^{max} > \lfloor cost/w_{min} \rfloor$ **then**
36: $I^{max} \leftarrow \lfloor cost/w_{min} \rfloor$
37: **end if**
38: **end if**
39: **end while**
40: $ctl.close()$
41: **return** $model^*$

TJ 符号化 (リスト 2) と TS 符号化 (リスト 3) との違いは，複数の重み付き遷移制約 (8–12 行目) と遷移コストを最小化するための目的関数 (14 行目) である．8 行目のルールは，各ステップ `t` に対して，遷移制約 TJ と遷移制約 TS が適用されたことを意味する補助アトム `tj(t)` と `ts(t)` をそれぞれ導入し，個数制約を用いていずれか一つが適用されることを強制している．10 行目のルールは TJ 遷移制約を表し，TJ 符号化 (リスト 2) の 9 行目に対応している．11–12 行目のルールは TS 遷移制約を表し，TS 符号化 (リスト 3) の 9–10 行目に対応している．`tj(t)` と `ts(t)` はどちらか必ず真になるので，10 行目と 11 行目は 2 つまとめて，`:- not 1 token_removed(X,t) 1.` と書いてもよい．14 行目のルールは，`#minimize` 文を

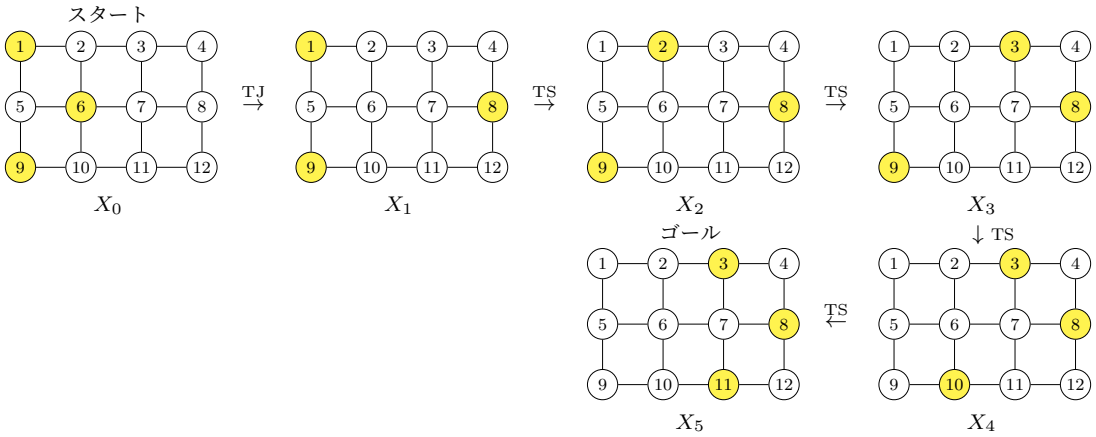

図 4　多目的独立集合遷移最適化問題の例

用いて，遷移コストを最小化している．この文の括弧中に出現する `tj_weight` と `ts_weight` は，それぞれ TJ と TS の重みを表す定数である．

```
#program base.
:- not in(X,0), start(X).

#program step(t).
K { in(X,t): node(X) } K :- k(K).
:- in(X,t), in(Y,t), edge(X,Y).

1 { tj(t) ; ts(t) } 1 :- t>0.
token_removed(X,t) :- in(X,t-1), not in(X,t), t>0.
:- tj(t), not 1 { token_removed(X,t) } 1.
:- ts(t), not 1 { token_removed(X,t) } 1.
:- ts(t), not 1 { in(Y,t) : edge(X,Y) ; in(Y,t) : edge(Y,X) } 1,
       token_removed(X,t).

#minimize { tj_weight,t : tj(t) ; ts_weight,t : ts(t) }.

#program check(t).
:- not in(X,t), goal(X), query(t).
```

リスト 4　多目的独立集合遷移最適化問題の ASP 符号化 (TJ–TS 符号化)

7　実行実験

提案解法の有効性を評価するために実験を行った．ベンチマークには多目的独立集合遷移最適化問題 (第 6 節) を使用した．ベンチマーク問題集には，国際組合せ遷移競技会 CoRe Challenge 2022 で使用された独立集合遷移の問題インスタンス (全 369 問) †1 を用いた．ただし，競技会において，TJ 型独立集合遷移での到達不能性が証明されたインスタンス 43 問は除外した．その理由は，TJ 遷移制約において到達不能であるとき，TS 遷移制約においても到達不能であるため，TJ と TS を考慮した独立集合遷移最適化においても到達不能となるためである．

多目的独立集合遷移最適化問題を解く ASP 符号化には，TJ–TS 符号化 (リスト 4) を用い，TJ と TS の重み (`tj_weight` と `ts_weight`) は，それぞれ 5 と 2 とした．多目的独立集合遷移最適化問題を解くソルバーには，提案アルゴリズム (アルゴリズム 1) を実装した ASP に基づく組合せ遷移最適化ソルバー *recongo_opt* を使用した．*recongo_opt* ソルバーのバックエンドには，ASP システム *clingo* バージョン 5.5.2 を使用した．1 問あたりの制限時間は 30 分とした．実験環境は Mac mini, 3.2GHz 6 コア Intel Core i7, 64GB メモリである．

解けた問題数を表 1 に示す．左から問題ファミリ名，問題ファミリに含まれる問題数，パレート最適な遷移系列が全列挙できた問題数 (#PF Found)，パレート最適な遷移系列が一つも得られなかった問題数 (#Unsolved) を表す．#PF Found はパレートフロントを得られた問題数を意味する．

提案手法は 326 問中 212 問 (全体の約 65%) に対

†1　`https://github.com/core-challenge/2022benchmark`

表 1　解けた問題数

問題ファミリ	問題数	#PF Found	#Unsolved
color04	202	187	15
grid	7	2	5
handcraft	5	5	0
power	17	0	17
queen	48	15	27
sp	30	3	27
square	17	0	17
Total	326	212	108

して，パレートフロントを得ることができた．問題ファミリごとに見ると，最短遷移長が短い問題の多い color04 に属する問題を 202 問中 187 問解き，よい性能を示している．その一方で，最短経路長が数百を超える問題を多く含む power, sp, square では，ほとんどの問題を解けなかった．クイーングラフ上の問題から構成される queen(計 48 問) では，パレートフロントを得られた問題が 15 問，パレート最適な遷移系列が一つも得られなかった問題が 27 問で，残り 6 問については，パレート最適な遷移系列を部分的に列挙することができた．

パレート最適な遷移系列を全列挙するのに要した CPU 時間を表 2 に示す．CPU 時間が 10 秒未満のインスタンスは省略している．最も時間を要したのは，頂点数が 400 で辺の数が 12540 のクイーングラフに基づくインスタンス queen020x020_04_0961 で，CPU 時間は約 1,630 秒であった．パレートフロントが得られた問題のうち，パレート最適な遷移系列が最も多かった問題は queen16_16_02 と queen060x060_03_1394 で，その数は 4 であった．

今回の実験で使用した多目的独立集合遷移最適化問題は TJ と TS のトレードオフを考慮した多様な解が得られるという長所がある．その一方で，遷移制約の数が 2 つと少なく，さらに TS が TJ の特殊形であるため，パレートフロントを構成する最適遷移系列の数が少ない結果となった．より多くの遷移制約を含む多目的独立集合遷移最適化問題を用いた性能評価は今後の重要な研究課題である．

8　関連研究

組合せ遷移．組合せ遷移の実践的研究は，ここ数年で活発に行われている．2022–2023 年に 2 年連続で開催された国際組合せ遷移競技会 CoRe Challenge 2022–2023 [26] では，独立集合遷移問題を解くソルバーの性能が競われた．この CoRe Challenge 競技会を契機として，組合せ遷移問題の汎用ソルバーがいくつか開発されている．*recongo* [33] は ASP に基づく汎用ソルバーであり，CoRe Challenge 競技会において，逐次ソルバーの複数部門で優勝するなど優れた性能を示している．他にも，並列ソルバーの複数部門で優勝した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* [18] などがある．しかし，これらのソルバーでは多目的組合せ遷移最適化問題を解くことはできない．本稿で開発した *recongo_opt* ソルバーは，第 5 節の提案アルゴリズムをもとに，*recongo* を多目的組合せ遷移最適化問題に拡張したものである．他のソルバーも多目的組合せ遷移最適化に拡張することは可能であると思われるが，そのためのアルゴリズムの改良・拡張は自明ではない．

個々の組合せ遷移問題の解法についても研究が進んでいる．組合せ遷移に対して ASP が初めて使われたのは，TJ 型独立集合遷移問題の ASP 符号化に関する研究である [32] [33]．文献 [15] はハミルトン閉路遷移問題を解く ASP 符号化を提案している．この研究では，遷移制約として，1 回の遷移でハミルトン閉路上のちょうど k 本の辺が変化する k-opt [29] が使用されている．ASP を用いた支配集合遷移問題の解法は，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，1 回の遷移でちょうど 1 個のトークンが追加あるいは削除される TAR 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．本稿の結果から，これらの代表的な組合せ遷移問題を多目的組合せ遷移最適化へ拡張することは興味深い．特に，ハミルトン閉路遷移問題の k-opt 遷移制約は，巡回セールスマン問題に対する近似解法でよく使われるヒューリスティックスであり，k-opt を使ったハミルトン閉路の多目的遷移最適化は今後の研

表 2　パレートフロントを求めるのに要した CPU 時間

インスタンス	CPU 時間 (秒)	インスタンス	CPU 時間 (秒)
hc-power-12_01	88.451	queen020x020_04_0961	1630.182
queen040x040_03_7037	417.181	queen050x050_03_0562	47.341
queen050x050_04_9047	142.805	queen060x060_01_8336	136.225
queen060x060_02_6710	150.821	queen060x060_03_1394	235.842
queen060x060_04_3799	277.343	3-FullIns_5_02	26.993
4-FullIns_5_01	31.045	4-FullIns_5_02	30.962
DSJC1000.5_01	18.403	DSJC500.9_02	14.184
DSJR500.1_02	30.773	le450_25a_02	66.956
le450_25b_02	47.521	mug100_25_02	522.049
mug88_1_02	70.956	qg.order30_02	21.79
qg.order60_01	616.283	queen13_13_01	15.809
school1_02	11.875	sp003_01	121.477
wap01a_01	95.606	wap02a_01	117.579
wap03a_01	199.054	wap04a_01	491.538
wap05a_01	11.537	wap06a_01	13.262
wap07a_01	25.297	wap08a_01	38.899

究課題の一つである．

プランニング． 自動プランニング [27] は，スタート状態からゴール状態への状態遷移系列に関する研究という点から，組合せ遷移と関連が深い研究分野である．例えば，最小コストプランニングは，状態を表す命題の集合，スタート状態，ゴール条件，コスト付きアクションの集合が与えられたとき，スタート状態からゴール条件を満たす状態に遷移するまでのアクション系列のうち，最小コストのものを求める問題である．最小コストプランニングは，2008 年から国際プランニング競技会 (International Planning Competition; IPC) の題材として使用され，実践的なプランナーの開発が活発に行われている．

最小コストプランニングは，アクション系列のコストを最小化することが目的であるため，プラン長 (すなわち，遷移長) は基本的に考慮しない．プラン長とコストを考慮したプランニングの研究としては文献 [22] がある．この研究では，制限されたプラン長での最小コストプランを求める問題を SAT に翻訳し，CDCL ベースの分枝限定法を用いたアルゴリズムによって解く手法を提案している．つまり，プラン長を 1 からインクリメンタルに増やしながら繰り返すことで，最短プラン長における最小コストプランを求める．また，文献 [7] は，アクションコストを加味した宣言的プランニング言語を新たに提案し，それを ASP に変換した後にプランを探索する手法を提案している．この手法は，最短プラン長における最小コストプランの他に，最小コストにおける最短プランを求めることができる．いずれの手法もパレート最適なプランを一つ見つけることはできるが，パレートフロントを求めることはできない．本稿の提案解法は，遷移長の上限値を自動で適切に更新することによって，パレート最適な遷移系列の全列挙を保証できる点が特長であり，プランニング分野への応用も期待できる．

9　おわりに

本稿では，遷移コストと遷移長を考慮したパレート最適な組合せ遷移を提案し，解集合プログラミングを用いた解法アプローチについて述べた．

今後の課題としては，多目的組合せ遷移最適化問題における解空間グラフの構造や性質に関する理論研究が挙げられる．本稿の提案手法の応用研究としては，

グラフ彩色遷移，$L(h,k)$ ラベリング遷移，ハミルトン閉路遷移の多目的組合せ遷移最適化への拡張を考えている．頂点間の距離を考慮した d-Jump 遷移制約 [14] に基づく多目的独立集合遷移最適化の研究も重要な研究課題の一つである．

参 考 文 献

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

[2] Bonamy, M., Dorbec, P., and Ouvrard, P.: Dominating sets reconfiguration under token sliding, *Discrete Applied Mathematics*, Vol. 301(2021), pp. 6–18.

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

[4] Brewster, R. C., McGuinness, S., Moore, B. R., and Noel, J. A.: A dichotomy theorem for circular colouring reconfiguration, *Theoretical Computer Science*, Vol. 639(2016), pp. 1–13.

[5] Cereceda, L., van den Heuvel, J., and Johnson, M.: Finding paths between 3-colorings, *Journal of Graph Theory*, Vol. 67, No. 1(2011), pp. 69–82.

[6] Christen, R., Eriksson, S., Katz, M., Muise, C., Petrov, A., Pommerening, F., Seipp, J., Sievers, S., and Speck, D.: PARIS: Planning Algorithms for Reconfiguring Independent Sets, *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, Gal, K., Nowé, A., Nalepa, G. J., Fairstein, R., and Radulescu, R.(eds.), Frontiers in Artificial Intelligence and Applications, Vol. 372, IOS Press, 2023, pp. 453–460.

[7] Eiter, T., Faber, W., Leone, N., Pfeifer, G., and Polleres, A.: Answer Set Planning Under Action Costs, *Journal of Artificial Intelligence Research*, Vol. 19(2003), pp. 25–71.

[8] Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T.: *Answer Set Solving in Practice*, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers, 2012.

[9] Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T.: Multi-shot ASP solving with clingo, *Theory and Practice of Logic Programming*, Vol. 19, No. 1(2019), pp. 27–82.

[10] 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.

[11] Gopalan, P., Kolaitis, P. G., Maneva, E. N., and Papadimitriou, C. H.: The Connectivity of Boolean Satisfiability: Computational and Structural Dichotomies, *SIAM Journal on Computing*, Vol. 38, No. 6(2009), pp. 2330–2355.

[12] Haas, R. and Seyffarth, K.: The k-Dominating Graph, *Graphs and Combinatorics*, Vol. 30, No. 3(2014), pp. 609–617.

[13] Haddadan, A., Ito, T., Mouawad, A. E., Nishimura, N., Ono, H., Suzuki, A., and Tebbal, Y.: The complexity of dominating set reconfiguration, *Theoretical Computer Science*, Vol. 651(2016), pp. 37–49.

[14] Hatano, H., Kitamura, N., Izumi, T., Ito, T., and Masuzawa, T.: Independent Set Reconfiguration Under Bounded-Hop Token, *CoRR*, Vol. abs/2407.11768(2024).

[15] Hirate, T., Banbara, M., Inoue, K., Lu, X., Nabeshima, H., Schaub, T., Soh, T., and Tamura, N.: Hamiltonian Cycle Reconfiguration with Answer Set Programming, *Proceedings of the 18th Edition of the European Conference on Logics in Artificial Intelligence (JELIA 2023)*, Gaggl, S. A., Martinez, M. V., and Ortiz, M.(eds.), LNCS, Vol. 14281, Springer, 2023, pp. 262–277.

[16] 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.

[17] Ito, T., Demaine, E. D., Harvey, N. J. A., Papadimitriou, C. H., Sideri, M., Uehara, R., and Uno, Y.: On the complexity of reconfiguration problems, *Theoretical Computer Science*, Vol. 412, No. 12-14(2011), pp. 1054–1065.

[18] Ito, T., Kawahara, J., Nakahata, Y., Soh, T., Suzuki, A., Teruyama, J., and Toda, T.: ZDD-Based Algorithmic Framework for Solving Shortest Reconfiguration Problems, *Proceedings of the 20th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2023)*, Ciré, A. A.(ed.), LNCS, Vol. 13884, Springer, 2023, pp. 167–183.

[19] Ito, T., Ono, H., and Otachi, Y.: Reconfiguration of Cliques in a Graph, *Proceedings of the 12th Annual Conference on Theory and Applications of Models of Computation (TAMC 2015)*, Jain, R., Jain, S., and Stephan, F.(eds.), LNCS, Vol. 9076, Springer, 2015, pp. 212–223.

[20] Kaminski, M., Medvedev, P., and Milanic, M.: Complexity of independent set reconfigurability problems, *Theoretical Computer Science*, Vol. 439(2012), pp. 9–15.

[21] Kato, M., Schaub, T., Soh, T., Tamura, N., and Banbara, M.: Dominating Set Reconfiguration with Answer Set Programming, *ICLP 2024, to appear.*

[22] Lu, Q., Huang, R., Chen, Y., Xu, Y., Zhang, W., and Chen, G.: A SAT-based approach to cost-sensitive temporally expressive planning, *ACM Transactions on Intelligent Systems and Technology*, Vol. 5, No. 1(2013), pp. 18:1–18:35.

[23] Mouawad, A. E., Nishimura, N., Pathak, V., and Raman, V.: Shortest Reconfiguration Paths in the Solution Space of Boolean Formulas, *SIAM Journal on Discrete Mathematics*, Vol. 31, No. 3(2017), pp. 2185–2200.

[24] 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.

[25] Nishimura, N.: Introduction to Reconfiguration, *Algorithms*, Vol. 11, No. 4(2018), pp. 52.

[26] Soh, T., Tanjo, T., Okamoto, Y., and Ito, T.: CoRe Challenge 2022/2023: Empirical Evaluations for Independent Set Reconfiguration Problems (Extended Abstract), *Proceedings of the Seventeenth International Symposium on Combinatorial Search (SOCS 2024)*, Felner, A. and Li, J.(eds.), AAAI Press, 2024, pp. 285–286.

[27] Son, T. C., Pontelli, E., Balduccini, M., and Schaub, T.: Answer Set Planning: A Survey, *Theory and Practice of Logic Programming*, Vol. 23, No. 1(2023), pp. 226–298.

[28] Suzuki, A., Mouawad, A. E., and Nishimura, N.: Reconfiguration of dominating sets, *Journal of Combinatorial Optimization*, Vol. 32, No. 4(2016), pp. 1182–1195.

[29] Takaoka, A.: Complexity of Hamiltonian Cycle Reconfiguration, *Algorithms*, Vol. 11, No. 9(2018), pp. 140.

[30] Toda, T., TakehiroIto, Kawahara, J., Soh, T., Suzuki, A., and Teruyama, J.: Solving Reconfiguration Problems of First-Order Expressible Properties of Graph Vertices with Boolean Satisfiability, *Proceedings of the IEEE 35th International Conference on Tools with Artificial Intelligence (ICTAI 2023)*, 2023, pp. 294–302.

[31] van den Heuvel, J.: The complexity of change, *Surveys in Combinatorics 2013*, Blackburn, S. R., Gerke, S., and Wildon, M.(eds.), London Mathematical Society Lecture Note Series, Vol. 409, Cambridge University Press, 2013, pp. 127–160.

[32] Yamada, Y., Banbara, M., Inoue, K., and Schaub, T.: Recongo: Bounded Combinatorial Reconfiguration with Answer Set Programming, *Proceedings of the 18th Edition of the European Conference on Logics in Artificial Intelligence (JELIA 2023)*, Gaggl, S. A., Martinez, M. V., and Ortiz, M.(eds.), LNCS, Vol. 14281, Springer, 2023, pp. 278–286.

[33] Yamada, Y., Banbara, M., Inoue, K., Schaub, T., and Uehara, R.: Combinatorial Reconfiguration with Answer Set Programming: Algorithms, Encodings, and Empirical Analysis, *Proceedings of the 18th International Conference and Workshops on Algorithms and Computation (WALCOM 2024)*, Uehara, R., Yamanaka, K., and Yen, H.(eds.), Lecture Notes in Computer Science, Vol. 14549, Springer, 2024, pp. 242–256.