Reconfillion
—組合せ遷移ソルバー—

川原　純

本稿では，組合せ遷移問題をゼロサプレス型二分決定グラフ (ZDD) を用いて解く手法を解説する．例として独立集合遷移問題を ZDD を用いて解く方法を紹介し，同じ枠組みで，全域木遷移やマッチング遷移など，多くの遷移問題を扱えることを示す．遷移のルールは複数存在するが，本手法はその中のいくつかのルールに対応している．応用を意識した例として，電力供給遷移問題を紹介する．筆者のグループが開発している組合せ遷移ソルバーの Python インターフェースである reconfillion と，GUI 操作により組合せ遷移問題を解くソフトウェア CoReViewer も紹介する．

キーワード：組合せ遷移，二分決定グラフ，グラフアルゴリズム

1. はじめに

科研費学術変革 (B) 「組合せ遷移の展開に向けた計算機科学・工学・数学によるアプローチの融合」の B01 班では，工学的なアプローチによる組合せ遷移の研究を実施してきた．配電網の切替手順を求める具体的な問題を足がかりに，さまざまな問題を解く汎用的なソルバーの開発を行ってきた．本稿では，複数開発したソルバーの一つである，ゼロサプレス型二分決定グラフ (Zero-suppressed binary decision diagram; ZDD) [1] と呼ばれるデータ構造を用いた，組合せ遷移ソルバー [2] を紹介する．

はじめに ZDD の概要を説明し，独立集合遷移問題を例として ZDD を用いたアルゴリズムの概要を解説する．このアルゴリズムが，独立集合遷移問題に限らず，多くの組合せ遷移問題を扱う枠組みとなることを示す．この枠組みで扱える遷移のルールについても述べる．応用を意識した例として，電力供給遷移問題を紹介し，本枠組みによってこの問題が扱えることを示す．

本手法は C++ 言語で実装して公開している．また，graphillion と呼ばれる，与えられたグラフのさまざまな部分グラフを列挙するライブラリと連携して，多くの組合せ遷移問題を解くことができる Python インターフェース reconfillion を開発中である．GUI から操作して組合せ遷移問題を解くことができるソフトウェア CoReViewer も紹介する．

かわはら　じゅん
京都大学大学院情報学研究科
〒 606–8501 京都市左京区吉田本町
jkawahara@i.kyoto-u.ac.jp

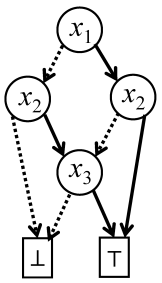

図 1　ZDD の例

2. ZDD

ZDD は集合族をコンパクトに圧縮して保持するデータ構造である．たとえば，集合族 $\{\{x_1,x_2\},\{x_1,x_3\},\{x_2,x_3\}\}$ を表す ZDD は図 1 となる．ZDD の正確な定義や詳細な説明は書籍 [3] にゆずり，ここでは 図 1 の ZDD の読み方だけ説明する．根節点（図 1 の x_1 と書かれた円）から ⊤ 終端（⊤ と書かれた長方形）までの経路を考える．たとえば，$x_1 \dashrightarrow x_2 \to x_3 \to \top$ という経路は，集合 $\{x_2,x_3\}$ に対応する（$\to$ の始点の要素を集合に含め，$\dashrightarrow$ の始点の要素は集合に含めない）．根節点から ⊤ 終端までの各経路と，集合族の各集合が一対一に対応する．

集合族の台集合 $U=\{x_1,\ldots,x_n\}$ は ZDD を構築する前に固定しなければならない．また，台集合の要素に順序（たとえば $x_1<\cdots<x_n$）を定める必要がある．x_i と書かれた節点が x_j と書かれた節点を指すとき，$x_i<x_j$ が成り立つ．

ZDD は集合族上の演算（和集合，共通部分，差集合，上位集合など）や，要素のランダムサンプリング，

フィルタリングなどを効率的に行える特徴がある [3]．これらの特徴を利用した例として，選挙区割問題 [4] やホットスポット検出 [5]，SNS の影響拡散の解析 [6] などが挙げられる．同様に本稿で紹介する組合せ遷移ソルバーもこれらの特徴を活用している．

3. ZDD を用いた組合せ遷移ソルバー

3.1 アルゴリズム

独立集合遷移問題 [7] を例に，組合せ遷移ソルバーの動作を説明する．グラフ $G=(V,E)$ の頂点集合 $V'\subseteq V$ は，任意の辺 $(u,v)\in E$ について，u と v の少なくとも片方が V' に含まれないとき，G の独立集合という．独立集合遷移問題では，$G=(V,E)$ と，二つの独立集合 $S\subseteq V$, $T\subseteq V$ が与えられたとき，S から T までの**遷移列**が存在するかを判定する問題である．ここで，S から T までの遷移列とは，$I_0=S, I_1, \ldots, I_{\ell-1}, I_\ell=T$ であり，各 $i=0,\ldots,\ell$ について，I_i は G の独立集合であり，各 $i=0,\ldots,\ell-1$ について，I_{i+1} は I_i に頂点を一つ追加し，別の頂点を一つ削除して得られる集合である（この遷移ルールはトークンジャンプと呼ばれる）．

ZDD を用いて独立集合遷移問題を解くことを考える．独立集合は頂点の集合であり，独立集合の集合は頂点の集合族として表される．集合族の台集合は V である．後々の一般化のため $U=V$ とし，以下では台集合を表す記号として U を用いる．頂点の順序を一つ固定する（頂点の順序はアルゴリズムの効率に大きく影響するが，本稿では述べない）．G のすべての独立集合の族を表現する BDD（Binary decision diagram; ZDD の変種）を効率的に構築するアルゴリズム [8] が存在し，このアルゴリズムを修正して ZDD を構築することは容易である．すなわち，ZDD の構築によって，G のすべての独立集合は（非明示的とはいえ）保持できているので，これを遷移問題の求解に用いることを考えたい．G のすべての独立集合の族を $\mathcal{I}_{\mathrm{ind}}$ とする．

論文 [2] では，ZDD で表された集合族 $\mathcal{F}$ に対し，$\mathcal{F}$ の各要素に対し，頂点を一つ追加し，別の頂点を一つ削除する操作で得られる集合族を表す ZDD を構築する方法を提案している．この集合族を $\mathsf{swap}(\mathcal{F})$ と書く．すなわち

$$\mathsf{swap}(\mathcal{F})=\{X\setminus\{x\}\cup\{y\} \mid X\in\mathcal{F}, x\in X, y\in U\setminus X\}$$

である．これを用いると，独立集合の族 $\mathcal{F}$ の各要素に対して，頂点の追加削除の操作が一気に可能となる．ただし，操作によって得られた集合は独立集合であるとは限らない．独立集合だけを得るために，G のすべての独立集合の族 $\mathcal{I}_{\mathrm{ind}}$ を用いる．$\mathsf{swap}(\mathcal{F})\cap\mathcal{I}_{\mathrm{ind}}$ を計算すれば，操作後に独立集合となるものだけを抽出できる．この計算は集合族の共通部分演算であり，$\mathsf{swap}(\mathcal{F})$ と $\mathcal{I}_{\mathrm{ind}}$ がともに ZDD として表されているので，$\mathsf{swap}(\mathcal{F})\cap\mathcal{I}_{\mathrm{ind}}$ も ZDD として構築可能である．

以上で述べた演算を利用して，独立集合遷移問題における S から T までの遷移列の存在判定を次の通りに行う．$\mathcal{F}_i$ を，S から 遷移ルール（頂点の一つ追加，一つ削除）を高々 i 回適用して得られる独立集合の族とする（途中の集合も独立集合でなければならない）．$\mathcal{F}_0=\{S\}$ とする．上記の議論から，$i=1,2,\ldots$ について，$\mathcal{F}_i=\mathsf{swap}(\mathcal{F}_{i-1})\cap\mathcal{I}_{\mathrm{ind}}$ が成り立つ．$\mathcal{F}_i$ が T を含むかどうか判定し，$\mathcal{F}_i$ が T を含むなら，S から T への遷移列が存在することがわかる．本手法では，$\mathcal{F}_i$ として遷移ルールを高々 i 回適用して得られる独立集合をすべて求めているので，$\mathcal{F}_i$ が T を含む最小の i が，S から T までの最短遷移列の長さである．さらに，$\mathcal{F}_i$ が T を含む i が存在せず，$\mathcal{F}_i=\emptyset$ となった時点で，S から T までの最短遷移列が存在しないことが判定できる．この方法は，S からの幅優先的な探索に相当する．ZDD の圧縮効果により，単純な幅優先探索よりも効率のよい探索が実現できている．

3.2 遷移対象となるグラフクラス

以上で述べた枠組みでは，独立集合特有の性質を用いているのは G のすべての独立集合族 $\mathcal{I}_{\mathrm{ind}}$（を表す ZDD）の部分だけである．一般的に，遷移問題に対して，その実行可能解の集合を ZDD として表現できれば，要素の一つ追加と一つ削除の遷移ルールの下で解くことができる．

さまざまな対象に対して，対象の集合を表す ZDD を効率的に構築できることが知られている．与えられたグラフ $G=(V,E)$ に対し，G の以下の対象の集合を表す ZDD を構築できる．

- 独立集合，クリーク，頂点被覆，支配集合

与えられたグラフ $G=(V,E)$ に対し E の部分集合（による辺誘導部分グラフ）による ZDD を構築できる．たとえば，マッチングや全域木などは E の部分集合として表すことができる．与えられたグラフ $G=(V,E)$ に対して，すべてのマッチングの族を ZDD として表す方法や，すべての全域木の族を ZDD として表す方法が知られている [9]．筆者は以前から，与えられたグラフに対して，さまざまな種類の部分グラフの族を

ZDD として表して，各種の問題に適用する研究をしている．以下の条件を組み合わせて記述された部分グラフの族を ZDD として表すことが可能である [9, 10].

- 各頂点の次数
  - 特定の頂点の次数を指定可
  - 特定の次数をもつ頂点の個数を指定可 [11]
- 各頂点の連結性
  - 特定の 2 頂点を必ず同じ連結成分に含める指定が可能
  - 特定の 2 頂点を必ず異なる連結成分に含める指定が可能
- 連結成分の個数（孤立した頂点を含めることも含めないことも可能）
- サイクルの有無
- 辺に重みを付与し，辺重みの総和が指定した重み以上/以下となる制約
- 部分グラフの族 $\mathcal{Z}'$ が ZDD として与えられたとき，
  - $\mathcal{Z}'$ のある要素を部分グラフとして含む
  - $\mathcal{Z}'$ のどの要素も部分グラフとして含まない

たとえば，「s,$t \in V$ の次数が 1，それ以外の頂点次数が 0 か 2，次数が 1 以上の全頂点が連結」という条件で s-t パス，「次数が 1 の頂点の個数が 2 個，残りの頂点の次数が 2」で（始点終点が任意の）ハミルトンパス，「すべての頂点の次数が 1」で完全マッチング，奇数長サイクルの族 $\mathcal{Z}'$ を与えて，「$\mathcal{Z}'$ のどの要素も部分グラフとして含まない」という条件で二部グラフを表せる．これらの条件で表せるグラフクラスとして，以下のものが挙げられる．

- 木，森，全域木，全域森，根付き全域森，シュタイナー木，パス，ハミルトンパス，サイクル，ハミルトンサイクル，マッチング，完全マッチング，k-正則グラフ，（次数が任意の）正則グラフ，次数指定部分グラフ，星グラフ，クリーク，二部グラフ

一部の種類については，組合せ遷移問題として意味をなさないものもある．たとえば，サイクル遷移問題では，サイクルの辺を一つ追加し，別の辺を一つ削除するとサイクルではなくなるので，遷移問題として意味をなさない．

さらに，最近の研究により，部分グラフの族 $\mathcal{Z}$ が ZDD として与えられたとき，$\mathcal{Z}$ の要素をいずれも誘導部分グラフとして含まない部分グラフの族を ZDD として構築する手法 [12] や，指定したグラフを位相的マイナーとして含まない部分グラフの族を ZDD として構築する手法 [13] も提案されている．これらの手法により，以下のグラフクラスを扱える．

- 弦グラフ：長さ 4 以上のサイクルは弦をもつ
- 弦二部グラフ：長さ 6 以上のサイクルは弦をもつ二部グラフ
- メニエルグラフ：長さ 5 以上の奇数長サイクルは弦を 2 本以上もつ
- d-爪自由グラフ (d-claw-free)：中央の次数が d である星グラフを誘導部分グラフとして含まない
- 偶穴（奇穴）自由グラフ (even/odd-hole-free)：偶数長（奇数長）サイクルを誘導部分グラフとして含まない
- 平面的グラフ：辺が交差しないように平面に描画できる
- 外平面的グラフ：すべての頂点が外面に面し，辺が交差しないように平面に描画できる
- 直並列グラフ：電気回路のように直列と並列の形からなる
- カクタス：すべての辺が高々一つのサイクルに含まれる

ZDD の演算を駆使することで，以下のグラフクラスも扱える [12].

- 区間グラフ：区間表現をもつ
- 真区間グラフ：任意の区間が別の区間を包含しない区間表現をもつ

これらも組合せ遷移問題として解くことが可能である．また，上記の条件を組み合わせることも可能である．たとえば「平面的グラフかつ区間グラフであり，指定した 2 点が連結であり，辺の本数が 10 本以上」という指定もできる．

3.3 遷移ルール

ZDD の演算としてどのような遷移が可能かについても研究を行っており，現在のところ，以下の遷移が可能であることがわかっている．

- swap：集合 $F \subseteq U$ から要素を一つ削除し，$U \setminus F$ の要素を一つ追加する．独立集合遷移問題のトークンジャンプルールに対応する．
- slide：集合 $F \subseteq U$ から要素を一つ削除し，$U \setminus F$ の要素を一つ追加する．ただし，追加・削除する要素は「隣接関係」にある必要がある．独立集合遷移問題のトークンスライディングルールに対応する．
- add/remove：集合 $F \subseteq U$ から要素を一つ削除するか，$U \setminus F$ の要素を F に一つ追加するか，どちらかを行う．要素数が指定した値以上/以下でなければならないという制約を課す．独立集合遷

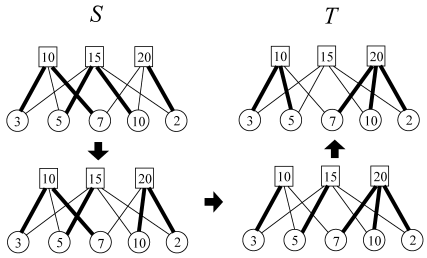

図 2　電力供給遷移問題

移問題のトークン追加削除ルールに対応する．

トークン追加削除ルールの「要素数が指定した値以上/以下」の制約は，解空間 ZDD を，その制約を満たすように構築することで与える．要素に重みを付与して，「要素の重み和が指定した値以上/以下」という制約を与えることも可能である．

4.　電力供給遷移問題

組合せ遷移問題のもう一つの例として，電力供給遷移問題 [7] を考える．電力供給問題は以下の問題である．図 2 のように二部グラフ $G = (V_1, V_2, E)$ が与えられる．二部グラフの一方の部 V_1 の頂点は供給点（図 2 の正方形の頂点）と呼ばれ，供給量と呼ばれる整数値をもつ．もう一方の部 V_2 の頂点は需要点（図 2 の円の頂点）と呼ばれ，需要量と呼ばれる整数値をもつ．各需要点は，ちょうど一つの供給点から供給されなければならない（図 2 の太線）．各供給点から供給されている需要点の需要量の総和は供給量以下でなければならない．すべての需要点が上記の条件を満たすように供給点から供給を受けているとき，供給に使用されている辺の集合を構成という．電力供給問題は，構成を求める問題である．

電力供給遷移問題は，二部グラフと各頂点の供給量，需要量，二つの構成 S, T が与えられたとき，S から T までの遷移列が存在するか判定する問題である．遷移ルールは，一つの需要点が供給される供給点の変更，すなわち，構成の一つの辺の追加と別の一つの辺の削除とする．遷移列の各構成は上記の条件を満たしていなければならない．

構成の集合を表す ZDD は以下のとおり構築できる．まず，各需要点について，その需要点に接続する辺のうち，ちょうど 1 本が使用されるという条件を表現する．頂点（需要点）v について，v に接続する辺の集合を $\delta(v)$ で表すとすると，「$\delta(v)$ の辺はちょうど 1 本使用される（$\delta(v)$ にない E の辺は使用されていてもされていなくてもよい）」という条件（需要点条件と呼ぶ）を満たす集合全体を $\mathcal{X}_v$ とすると，

$$\mathcal{X}_v = \{\{x\} \cup E' \mid x \in \delta(v), E' \subseteq E \setminus \delta(v)\}$$

である．詳細は省略するが，$\mathcal{X}_v$ を表す ZDD は構築可能である．したがって，すべての需要点について需要点条件を満たす集合族は $\bigcap_{v \in V_2} \mathcal{X}_v$ であり，これを表す ZDD も共通部分演算で構築可能である．

供給点についても，詳細は省略するが，同様に条件を満たす集合族を表す ZDD を構築できる．両者の共通部分が（すべての条件を満たす）構成の集合族 ZDD である．解集合 ZDD が構築できたので，3 節で述べた枠組みが適用可能である．

5.　ソフトウェア

5.1　組合せ遷移ソルバーの Python インターフェース reconfillion

C++ 言語で実装した ZDD を用いた組合せ遷移ソルバーを，ddreconf と名付けて，GitHub のページで公開している [14]．C++ 言語によるプログラムを動作させるには，ソースコードのビルド環境が必要なため，敷居が若干高い．そこで，Python 言語を用いた，組合せ遷移問題を解くためのインターフェースを開発中である．3.2 節で紹介したさまざまなグラフクラスに対応するため，graphillion [15] と呼ばれる，ZDD を用いたグラフ集合処理ライブラリと連携している．組合せ遷移の Python インターフェースを，graphillion の名前を一部借りて，reconfillion と名付けて公開している [16]．現時点では，reconfillion は独立集合遷移問題だけを扱うことができる．将来的には，reconfillion は graphillion で扱えるすべてのグラフクラスに対応する遷移問題を扱える予定である．Reconfillion のコード例を以下に示す．

```
# 3 x 3 grid の頂点集合
vertices = [1, 2, 3, 4, 5, 6, 7, 8, 9]
# 辺集合
edges = [(1, 2), (1, 4), (2, 3), (2, 5),
 (3, 6), (4, 5), (4, 7), (5, 6),
 (5, 8), (6, 9), (7, 8), (8, 9)]
# 台集合を設定
setset.set_universe(vertices)

# 独立集合全体を列挙（graphillion の機能を
# 用いて，内部では ZDD 表現されている）
iss = reconf.get_independent_setset(
 vertices, edges)
```


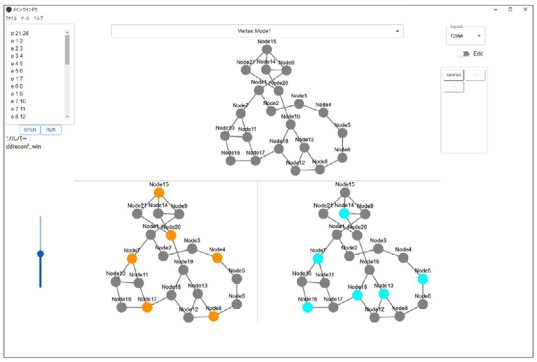

図 3　CoReViewer

```
s = {2, 4, 6} # 開始独立集合
t = {1, 6, 8} # 目標独立集合

# s から t への遷移列を求める
seq = reconf.get_reconf_seq(s, t, iss)

for x in seq: # 遷移列を print
 print(x)
```

5.2　GUI ビュアー CoReViewer

5.1 節で述べた reconfillion によって，Python に詳しい利用者は組合せ遷移問題の求解を試すことができるが，プログラミングをせずに組合せ遷移問題を GUI で操作して解くことのできるソフトウェアも作成して公開している．内部のソルバーエンジンは差し替え可能となっており，組合せ遷移問題の競技会である CoRe Challenge [17] の出力形式（DIMACS 形式）に対応したソルバーを利用することができる．GUI の部分を CoReViewer [18] と名付けた．スクリーンショットを図 3 に示す．

CoReViewer は JavaScript 言語で実装されており，マルチプラットフォーム対応の electron フレームワークを用いているため，Windows，Mac，Linux に対応している．現在のところ，CoReViewer はグラフに関する組合せ遷移問題に対応している．画面の上部にはグラフが表示されている．グラフは cytoscape.js ライブラリによって描画されており，頂点の移動や，頂点や辺の追加，削除など，（cytoscape.js ライブラリが提供する）基本的なグラフ編集機能を備えている．グラフの自動レイアウトにも対応しており，読み込んだグラフを見やすく描画できる．画面の下部は，開始と目標の配置が表示されている．こちらもマウスクリックによる編集が可能である．

上部のリストボックスから，解きたい組合せ遷移問題を選択でき，「Run」ボタンを押下することで，内部のソルバーエンジンが実行される．解（遷移列）が得られると，1 ステップずつ遷移対象を表示することができる．

6.　おわりに

ZDD を用いた組合せ遷移ソルバーを紹介した．さまざまな対象の解集合を ZDD として構築できることを利用した，組合せ遷移を解くための汎用的な枠組みを示し，電力供給問題が扱えることを示した．

ZDD ソルバー (ddreconf [14]) は，前述した競技会 CoRe Challenge 2022 [17] に参加したが，ほかの出場ソルバーのほとんどが解を求めることに成功した問題インスタンスに対しても，ZDD ソルバーは独立集合族を表す ZDD の構築ができず，遷移の 1 ステップも求めることができなかった．一方で，解の遷移列の遷移長が長くなるいくつかのインスタンスに対しては，ZDD ソルバーだけが解けており，ほかのソルバーは解けなかった．ほかのソルバーは，SAT や A*探索をベースとしており，ZDD ソルバーはそれらとは異なる傾向の挙動が見られた．解集合の ZDD は，対象のすべての情報を含んでいるため，圧縮されているとはいえ，ZDD の大きさは莫大になりがちである．ZDD ソルバーのさらなる性能改善は今後の課題である．

謝辞　組合せ遷移ソルバー Python インターフェースの開発メンバーである山崎宏紀氏（元京都大学）にソースコード例を書いていただきました．感謝いたします．本研究の一部は，JSPS 科研費 JP18H04091，JP20H05794 の助成を受けたものです．

参考文献

[1] S. Minato, “Zero-suppressed BDDs for set manipulation in combinatorial problems,” In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pp. 272–277, 1993.
[2] T. Ito, J. Kawahara, Y. Nakahata, T. Soh, A. Suzuki, J. Teruyama and T. Toda, “ZDD-based algorithmic framework for solving shortest reconfiguration problems,” In *Proceedings of the 20th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2023)*, pp. 167–183, 2023.
[3] ERATO 湊離散構造処理系プロジェクト（著），湊真一（編），『超高速グラフ列挙アルゴリズム―〈フカシギの数え方〉が拓く，組合せ問題への新アプローチ―』，森北出版，2015.

[4] J. Kawahara, T. Horiyama, K. Hotta and S. Minato, "Generating all patterns of graph partitions within a disparity bound," In *Proceedings of the 11th International Conference and Workshops on Algorithms and Computation (WALCOM2017)*, pp. 119–131, 2017.
[5] F. Ishioka, J. Kawahara, M. Mizuta, S. Minato and K. Kurihara, "Evaluation of hotspot cluster detection using spatial scan statistic based on exact counting," *Japanese Journal of Statistics and Data Science*, **2**(1), pp. 1–15, 2019.
[6] T. Maehara, H. Suzuki and M. Ishihata, "Exact computation of influence spread by binary decision diagrams," In *Proceedings of the 26th International Conference on World Wide Web*, pp. 947–956, 2017.
[7] T. Ito, E. D. Demaine, N. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara and Y. Uno, "On the complexity of reconfiguration problems," *Theoretical Computer Science* **412**, pp. 1054–1065, 2011.
[8] K. Hayase, K. Sadakane and S. Tani, "Output-size sensitiveness of OBDD construction through maximal independent set problem," In *Proceedings of Computing and Combinatorics (COCOON 1995)*, pp. 229–234, 1995.
[9] K. Sekine, H. Imai and S. Tani, "Computing the Tutte polynomial of a graph of moderate size," In *Proceedings of the 6th International Symposium on Algorithms and Computation (ISAAC 1995)*, pp. 224–233, 1995.
[10] J. Kawahara, T. Inoue, H. Iwashita and S. Minato, "Frontier-based search for enumerating all constrained subgraphs with compressed representation," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **E100-A**, pp. 1773–1784, 2017.
[11] J. Kawahara, T. Saitoh, H. Suzuki and R. Yoshinaka, "Solving the longest oneway-ticket problem and enumerating letter graphs by augmenting the two representative approaches with ZDDs," In *Proceedings of the Computational Intelligence in Information Systems Conference (CIIS 2016)*, pp. 294–305, 2016.
[12] J. Kawahara, T. Saitoh, H. Suzuki and R. Yoshinaka, "Colorful frontier-based search: Implicit enumeration of chordal and interval subgraphs," In *Proceedings of the Special Event on Analysis of Experimental Algorithms (SEA^2 2019)*, pp. 125–141, 2019.
[13] Y. Nakahata, J. Kawahara, T. Horiyama and S. Minato, "Implicit enumeration of topological-minor-embeddings and its application to planar subgraph enumeration," In *Proceedings of the 14th International Conference and Workshops on Algorithms and Computation (WALCOM 2020)*, pp. 211–222, 2020.
[14] J. Kawahara, ddreconf, https://github.com/junkawahara/ddreconf （2023 年 3 月 24 日閲覧）
[15] T. Inoue, graphillion, https://github.com/takemaru/graphillion （2023 年 3 月 24 日閲覧）
[16] J. Kawahara and H. Yamazaki, reconfillion, https://github.com/junkawahara/reconfillion （2023 年 3 月 24 日閲覧）
[17] T. Soh, Y. Okamoto and T. Ito, "Core challenge 2022: Solver and graph descriptions," *CoRR*, abs/2208.02495, 2022.
[18] J. Kawahara, CoReViewer, https://github.com/junkawahara/CoReViewer （2023 年 3 月 24 日閲覧）

