Vol.2018-AL-169 No.6
2018/9/3

SeqBDDを用いた集合分割の族の表現法と実験的評価

高橋 翔哉[1,a)]　湊 真一[2,b)]　瀧川 一学[1,c)]

概要：ある集合を重なりのない部分集合に分けたものを集合分割と呼ぶ．集合分割は選挙区割り問題やスケジューリング問題などの応用先を持ち，これらは集合分割の集合 (集合分割の族) に対する最適化問題として表現することが可能である．このように集合分割の族を考えることは重要であるが，集合分割の総数は集合の要素数に対して爆発的に増加するために扱うのが難しく，集合分割の族を扱う試みはあまりなされてこなかった．本研究では集合分割をある文字列で表現される操作 (操作列) で表し，その操作列の集合を Sequence Binary Decision Diagram(SeqBDD) と呼ばれる文字列集合を扱うデータ構造を用いて表現することで集合分割の族を扱う手法を提案する．本稿では様々な集合分割の族を SeqBDD で表現する手法とその計算機実験結果を示す．

1. はじめに

集合分割とはある集合を重なりのない部分集合に分けたもののことを指し，選挙区割り問題 [1] やスケジューリング問題 [2] などの集合分割の集合 (集合分割の族) を解空間とする組合せ最適化問題に応用可能である．このことから集合分割の族を扱うことは重要であると考えられるが，集合分割の総数は集合の要素数に対して爆発的に増加することが知られているため，その集合を扱うのは困難である．そのため，上記に挙げられる組合せ最適化問題においてある制約を満たした集合分割の族を解空間とすることで，一つの最適解もしくは近似解を求める手法は過去に提案されてきた [3]．しかしながら，実際に一つの最適解を得た後に，得られた解がある程度想定されたものではない (実際に解として使用することが困難である) ことや，要求される制約の変化によって最適解も変化することなど，求めたものとは別の解が必要になることがあると考える．そのような場合，最適解一つだけではなく他の解候補を複数持っておき需要の変化に対応可能にしておくこと，つまり集合分割の列挙および特定の集合分割だけ取り出す操作などを実行できるような索引化を行うことは有効となる．ただし，先述の通り集合分割の総数は膨大な数であるため，その列挙・索引化を行うためには適切なデータ構造を用いる必要がある．

一方で，ある離散構造を扱うデータ構造として Decision Diagram と呼ばれるものが存在する．例えば，Binary Decision Diagram(BDD) [4] は論理関数を扱うデータ構造として知られる．Decision Diagram は対象の離散構造をコンパクトに表現することに加え，特定の条件を満たしたものだけを取り出す演算や個々の離散構造特有の演算 (BDD ならば論理演算など) を圧縮を維持したまま実行可能であることが知られている．このことから Decision Diagram は離散構造を列挙・索引化するのに適したデータ構造の一つであるということができ，様々な離散構造を対象として近年盛んに研究されている [5,6]．以上から，このような特長を有する Decision Diagram を用いて離散構造の一つである集合分割の族を表現することで先述の列挙および索引化を達成可能であると考えられ，本研究の目的としている．

本稿では集合分割の族を Decision Diagram の一つである Sequence Binary Decision Diagram(SeqBDD) と呼ばれる文字列集合を扱うデータ構造を用いて表現する手法を提案する．提案手法は集合分割をある文字列で表される操作で表現し，その集合を SeqBDD で表現することによって集合分割の族を扱う．提案手法によって，集合分割の族をコンパクトに表現しやすく，ある制約を満たした集合分割のみを取り出す操作を SeqBDD 上の演算を用いることで可能となる．2 節では本研究の対象である集合分割および集合分割の族を定義し，基本的な性質について述べる．また，本研究で用いる SeqBDD およびそれと深く関係がある ZDD と呼ばれるデータ構造についてその性質・特徴を述べる．3 節では様々な集合分割の族を SeqBDD で表現する提案手法について述べ，4 節でその計算機実験結果を示す．最後に 5 節で本稿のまとめと今後の課題について述べる．

[1] 北海道大学 大学院 情報科学研究科
[2] 京都大学 大学院 情報学研究科
[a)] s_takahashi@ist.hokudai.ac.jp
[b)] minato@i.kyoto-u.ac.jp
[c)] takigawa@ist.hokudai.ac.jp

2. 準備

2.1 集合分割

まずは集合分割を定義する.

定義 2.1 (集合分割) ある単純集合 X の部分集合族 $\{C_1,\ldots,C_m\}$ が

- $\emptyset \notin C_i\,(i=1,...,m)$
- $C_1 \cup C_2 \cup \ldots \cup C_m = X$
- $C_i \cap C_j = \emptyset\,(i \neq j)$

を満たすとき $\{C_1,\ldots,C_m\}$ を X の集合分割と呼ぶ.

また，このような C_i について以下のように定義する.

定義 2.2 (セル) ある集合分割の $\{C_1,\ldots,C_m\}$ の $C_i\,(i=1,\ldots,m)$ をセルと呼ぶ.

例えば 3 要素の集合 $X=\{a,b,c\}$ に対する集合分割は，$\{\{a\},\{b\},\{c\}\},\{\{a,b\},\{c\}\},\{\{a,c\},\{b\}\},\{\{a\},\{b,c\}\}$ $\{\{a,b,c\}\}$ で総数は 5 となる. 本稿では簡単のため, それぞれ $(a\,|\,b\,|\,c),(a\,b\,|\,c),(a\,c\,|\,b),(a\,|\,b\,c),(a\,b\,c)$ のように集合分割を表記する. また，セルは $\{a\}$, $\{b\}$, $\{c\}$, $\{a,b\}$, $\{a,c\}$, $\{b,c\}$, $\{a,b,c\}$ の 7 つである.

n 要素の集合に対する集合分割の総数はベル数 [7] と呼ばれ，n 番目のベル数 B_n に対し，$B_0=B_1=1$ として，$B_n=\sum_{k=0}^{n-1}\binom{n-1}{k}B_k$ で表される. この式から集合分割の総数は n に対して爆発的に増加することが確認できる.

さらに，本研究で扱う集合分割の族について以下のように定義する.

定義 2.3 (集合分割の族) 集合分割を要素とする集合を集合分割の族と呼ぶ.

例えば 3 要素の集合 $X=\{a,b,c\}$ に対して，$\{(a\,b\,c)\}$ や $\{(a\,c\,|\,b),(a\,|\,b\,|\,c)\}$ などが集合分割の族の例として挙げられる. 集合分割の集合である集合分割の族は 2^{B_n} 通り存在することとなり，扱うのが困難である.

2.2 ZDD

Zero-suppressed Binary Decision Diagram(ZDD) [8] は集合族を表現する非巡回有向グラフである. ZDD は根節点と呼ばれる親を持たない節点が 1 つ，終端節点と呼ばれる子を持たない節点が 2 つ，それ以外が分岐節点と呼ばれる 2 つの出枝を持つ節点で構成される. 分岐節点には各レベル毎に設定された要素名がラベル付けされ，1-枝，0-枝と呼ばれる 2 つの出枝でその要素を選択するか否かを表現する. また，2 つの終端節点にはそれぞれ 1,0 がラベル付けされ，根節点から終端へのパスによって場合分けされた集合が，表現する集合族に含まれるか否かを表す. 例えば，$\mathcal{F}=\{\{a,b\},\{a,c\},\{c\}\}$ を表現する ZDD は図 1 のようになる.

ZDD は，和集合や共通集合をとる基本的な集合演算や特定の要素を含む集合族を取り出すなどの集合族に関する演算を圧縮を維持したまま実行できることが知られている.

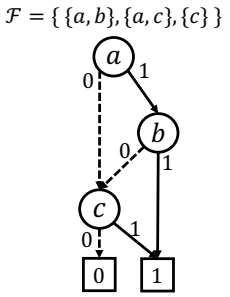


図 1: $\mathcal{F}$ を表現する ZDD

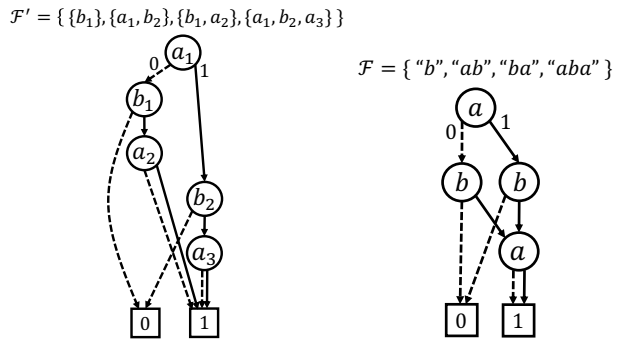
このことから ZDD を用いることで，扱う集合族について更に絞りこみを行ったり，複雑に表記される集合族を集合演算を介することで簡単に得ることができる.

文字列 (系列) 集合はテキスト処理やパターンマイニングなどの観点から，重要な離散構造の一つである. 文字列は各要素の出現位置の情報を保持した多重集合と考えることができるため，文字列集合をその多重集合の族として表現することが可能である. ZDD はそのままでは多重集合および要素の出現位置の情報を保持した集合を含むことは不可能であり，文字列集合を扱うことはできない. これは，ZDD 節点へのラベル付けの際に要素の出現位置を添字などとして情報をもたせておくことで可能となる. 例えば，文字列 “ab” ならば $\{a_1,b_2\}$，“ba” ならば $\{b_1,a_2\}$ というようにすれば文字列集合を集合族として表すことができ，ZDD で表現できることとなる. しかしながら，ZDD 変数の数が文字列に使用される文字の種類と文字列長の積に依存するため大きくなりやすいことに加え，要素の出現位置まで一致しなければ節点共有が行われないために圧縮が上手く働かない場合があるという問題があった. そこで，文字列集合を扱うのにより適したデータ構造として考案されたのが次節で述べる SeqBDD である.

2.3 SeqBDD

Sequence Binary Decision Diagram(SeqBDD) [9] は文字列集合を扱うのに特化させた ZDD の変化型である. 基本的な構成は ZDD と同様であるが変数順序に関しては異なる点が存在する. ZDD の分岐節点は与えられた変数順序に従い，自身にラベル付けされた要素名よりも前の要素名を持つ節点には枝を持つことができない. 一方 SeqBDD の分岐節点は 1-枝が指す節点についてのみ変数順序に従わないことを許容し，どの要素名を持つ節点にも 1-枝を持つことができる. これにより根節点から終端節点に至るある一つのパスが，同じ要素を複数持つことを許容した上で要素の出現位置の情報を保持した集合，すなわち文字列を表し，SeqBDD が文字列集合を表現することとなる. 例として文字列集合 $\mathcal{F}=\{$“b”, “ab”, “ba”, “aba”$\}$ を ZDD および SeqBDD で表現した例を図 2 に示す. この図からも SeqBDD が ZDD よりも文字列集合をコンパクトに表現可

能であることを確認できる．



(a) ZDD　　(b) SeqBDD

図 2: $\mathcal{F}$ を表現する (a)ZDD と (b)SeqBDD

SeqBDD では文字列集合に関する演算を圧縮を維持したままに実行可能であることが知られている．和集合演算や共通集合演算などの基本的な集合演算に加え，文字列集合の直積集合演算という拡張演算を扱うことができる．ここでの文字列集合の直積集合演算とは 2 つの文字列集合 $\mathcal{F}, \mathcal{G}$ に対して，文字列の結合を演算子 $+$ で表すこととすれば

$$\mathcal{F} * \mathcal{G} = \{f + g \mid f \in \mathcal{F}, g \in \mathcal{G}\}$$

のように表記される演算のことを指す．例えば $\mathcal{F} = \{\text{“}a\text{”}, \text{“}ab\text{”}\}, \mathcal{G} = \{\text{“}b\text{”}, \text{“}bc\text{”}\}$ の場合を考えると，$\mathcal{F} * \mathcal{G} = \{\text{“}ab\text{”}, \text{“}abc\text{”}, \text{“}abb\text{”}, \text{“}abbc\text{”}\}$ となる．この直積集合演算に関しても SeqBDD を用いることで圧縮を維持したままで実行することができる．

3. 提案手法

3.1 概要

集合分割における ‘|’ を一つの文字としてみなせば，任意の集合分割は文字列で表現できる．例えば $(\,a\,c\,|\,b\,)$ という集合分割は “$ac|b$” という文字列であるとみなすことができる．このことから集合分割の族は文字列集合で表すことが可能であり，SeqBDD で表現できることとなる．しかしながらこの表現による SeqBDD では，根節点から終端節点へのそれぞれのパスの途中でどの要素を文字列として含んでいるかを管理しなければならないため，目的の SeqBDD 構築が困難である場合が存在すると考えられる．

以上から，提案手法では SeqBDD 構築により適した集合分割の文字列表現を考え，それを基とした SeqBDD の構築法を与える．

3.2 集合分割の操作列表現

提案手法では n 要素の集合 X を入力として X のある一つの集合分割を出力とするような，文字列で表された操作 (操作列)L を考えることで集合分割を表現する．L は n 進数表記の数字 $i_{(n)}\,(1 \leq i < n)$ と特殊文字 ‘S’ を組み合わせることで構成される長さ n 未満の文字列であり，この L に対して以下のような操作を考える．

入力: n 要素の集合 $X = \{e_1, \ldots, e_n\}$
出力: X のある集合分割
A1. L の次の文字 M を読み込む
A2. M が数字 $i_{(n)}$ ならば X の $i'_{(10)}\,(i_{(k)} = i'_{(10)})$ 番目の要素を出力し，その要素を X から削除した後 A1. へ
A3. M が ‘S’ ならば X の最も大きい要素と ‘|’ を出力し，その要素を X から削除した後 A1. へ
A4. M が NULL ならば X の要素をすべて出力し，終了

例として “1S” という操作列に対して $X = \{a, b, c\}$ を入力することを考える．まず A1 で文字を読み込むと ‘1’ であるので，A2 の処理を行う．A2 において $i_{(n)} = 1 = i'_{(10)}$ であるので X の 1 番目の要素 a を出力し，X から a を削除する．再び A1 で次の文字を読み込むと ‘S’ であるので，A3 の処理を行う．A3 では X の最も大きい要素 c と ‘|’ を出力し，X から c を削除する．この時点で $(\,a\,c\,|\,)$ が出力されていることとなる．その後 A1 で次の文字を読み込もうとするが，終端まで読み込んでいるので A4 の処理を行う．A4 では削除されずに残った X の要素すべて，すなわち b を出力し，操作を終了する．以上から，結果として $(\,a\,c\,|\,b\,)$ という出力が得られる．また，空文字列 “ε” に対して同様の集合 X を入力すると，A1 で NULL が読み込まれるため A4 の処理，X の要素 a, b, c の出力を行う．これにより $(\,a\,b\,c\,)$ という出力が得られることとなる．このように X の集合分割 $(\,a\,|\,b\,|\,c\,), (\,a\,b\,|\,c\,), (\,a\,c\,|\,b\,), (\,a\,|\,b\,c\,), (\,a\,b\,c\,)$ に対して，セルが順不同であることを考慮するとそれぞれ “SS”, “S”, “1S”, “2S”, “ε” という操作列を対応付けることができる．

上述のようにある集合分割に対して操作列を対応させることができる．このことから，任意の集合分割の族に対して，含まれる集合分割に対応した操作列の集合 (操作列集合) を考えることで集合分割の族を扱うことが可能となる．操作列は文字列であるので，操作列集合は文字列集合となり SeqBDD を用いて表現可能である．この表現法による SeqBDD では根節点から終端節点までのパスの途中で，集合に残っているうちの何番目の要素を使用するかのみを考えるため，どの要素を使用したかを管理する必要はなく，扱いが容易になっていると考えられる．

3.3 操作列集合の生成と SeqBDD 構築

前節で操作列集合を SeqBDD で表現することによって集合分割の族を扱うことは可能になったが，この SeqBDD をどのようにして得るかが問題となる．方法の一つとして個々の操作列を表現する SeqBDD を構築し，それらの SeqBDD の和集合演算を行うことで目的のものを得る手法

が考えられるが，操作列集合の要素数は最大でベル数と等しくなるため，現実的な時間では実行不可能である．以上から，本節では目的の SeqBDD を効率よく構築できるような操作列集合の生成法を与え，それに従って SeqBDD 構築を行う手法を考える．本稿では，制約を考慮しない場合と何らかの制約を考慮した場合を考え，それぞれについて 3.3.1 節，3.3.2 節で述べる．

3.3.1 制約を考慮しない操作列集合

制約を考慮しない，つまり n 要素の任意の集合分割を含む集合分割の族に対応する操作列集合 $\mathcal{L}_n$ の生成について考える．ここで，ある操作列の文頭から最初に 'S' が出現するまでの部分操作列 "$k\,(0 \le k < n-1)$ 個の重複ない n 未満の数字" + "S" による挙動に注目する．この部分操作列に n 要素の集合 X を入力すると，A2 の処理によって k 個の要素が，A3 によって最も大きい要素と '|' が出力される．すなわち，この部分操作列は最も大きい要素とそれ以外の要素 k 個を含むセルを生成する操作であると考えることができる．上記部分操作列による操作終了後 A2 および A3 の処理によって X には $n-k-1$ 要素が残るため，その後は $n-k-1$ 要素の集合分割を出力する操作列を考えればよい．以上から，"$k\,(0 \le k < n-1)$ 個の重複ない n 未満の数字" + "S" + L ($\mathrm{L} \in \mathcal{L}_{n-k-1}$) によって n 要素の集合に対する操作列を生成することができる．例えば $n=3, k=0$ の場合，$n-k-1=2$ 要素の集合分割 $(a\,|\,b), (a\,b)$ とそれに対応した操作列 "S", "ε" の文頭に "0 個の n 未満の数字"+"S" を結合すると，$(a\,|\,b\,|\,c), (a\,b\,|\,c)$ と対応した "SS", "S" が生成される．同様に $n=3, k=1$ の場合，$n-k-1=1$ 要素の集合分割 (a) とそれに対応した操作列 "ε" の文頭に "1 個の n 未満の数字"+"S" を結合すると，$(a\,c\,|\,b), (a\,|\,b\,c)$ と対応した "1S", "2S" が生成されることとなる．

以上の生成法は文字列集合に関する演算のみを用いて表現することができる．ここで，n 以下の数字を重複なく降順に k 個並べた文字列の集合 $\mathrm{Comb}(n,k)$ とすれば，n 要素の操作列集合 $\mathcal{L}_n$ は

$$\mathcal{L}_n = \{\text{“}\varepsilon\text{”}\} \cup (\textstyle\bigcup_{k=0}^{n-2}(\mathrm{Comb}(n-1,k) * \{\text{“}S\text{”}\} * \mathcal{L}_{n-k-1}))$$

と書くことができる．上式の右項について，$\mathrm{Comb}(n,k)$ で n 番目の要素と同じセルに含まれる要素を選んだ後，"S" によって n 番目の要素を含むセルを生成する．このとき，$\mathrm{Comb}(n,k)$ が数字を降順に並べるという性質から，n 番目の要素と同じセルに含まれる要素の選び方に対応した文字列は一意に定まる．その後残った $n-k-1$ 要素の任意の操作列への結合を，L_{n-k-1} と文字列の直積集合演算で行うことで実行する．この演算で生成されない "ε" を和集合演算で追加することによって，目的の $\mathcal{L}_n$ が得られることとなる．

上記生成式が正しいことを確認するため，まず $\mathcal{L}_n$ の要素数が集合分割の総数と一致しているかを確認する．$\mathcal{L}_0 = \{\text{“}\varepsilon\text{”}\}$ とすれば

$$\begin{aligned}
|\mathcal{L}_n| &= 1 + \sum_{k=0}^{n-2}\left(\binom{n-1}{k}|\mathcal{L}_{n-k-1}|\right) \\
&= \sum_{k=0}^{n-1}\left(\binom{n-1}{k}|\mathcal{L}_{n-k-1}|\right) \quad \left(\because \binom{n-1}{n-1}|\mathcal{L}_0| = 1\right) \\
&= \sum_{k=0}^{n-1}\left(\binom{n-1}{k}|\mathcal{L}_k|\right) \qquad \left(\because \binom{n}{k} = \binom{n}{n-k}\right)
\end{aligned}$$

となる．この式は 2.1 節で示したベル数の導出式と一致するため，$|\mathcal{L}_n| = B_n$ であり，n 要素の集合分割の総数と一致していることが分かる．次に，生成されたある 2 つの操作列が同じ集合分割を出力することがないかを確認する．同じ集合分割と対応づけられる 2 つの操作列 L_1, L_2 の存在を仮定する．このとき，操作列において最初に出現する 'S' は必ず集合の最大の要素 e_n を含むセルを生成するため，L_1, L_2 それぞれの部分操作列によって生成されるセルは必ず等しくならなければならない．しかしながら $\mathrm{Comb}(n,k)$ の性質から，e_n と同じセルに含まれる他の要素の選び方は一意であるので，複数の部分操作列が同じセルを生成することはない．よって，生成される操作列には対応する唯一の集合分割が存在することが分かる．以上から，生成される操作列の数は n 要素の集合分割の総数と一致し，各操作列に対してただ一つの集合分割が対応付けられるので，生成される操作列と集合分割は一対一に対応していることが確認できる．

上記で導出した $\mathcal{L}_n$ の生成式を基とした SeqBDD 構築について考える．$\mathcal{L}_n$ の生成式は文字列集合に関する演算のみで構成されている．さらに，用いられている演算は和集合演算および直積集合演算のみであり，SeqBDD で実行可能な演算である．また，$\mathrm{Comb}(n,k)$ についても $\mathrm{Comb}(n-1,k)$, $\mathrm{Comb}(n-1,k-1)$ を用いて

$$\mathrm{Comb}(n,k) = \mathrm{Comb}(n-1,k) \cup (\{\text{“}n\text{”}\} * \mathrm{Comb}(n-1,k-1))$$

のように書くことができ，文字列の和集合演算および直積集合演算のみを用いて表現可能であることが分かる．以上から，SeqBDD で実行可能な演算のみを用いて $\mathcal{L}_n$ を表現する SeqBDD を構築可能であることが分かり，構築アルゴリズムを以下のように書くことができる．

Algorithm 1 $\mathcal{L}_n$ を表現する SeqBDD 構築

Input: $n \in \mathbb{N}$
Output: $\mathcal{L}_n$ を表現する SeqBDD F_n

SeqBDD $F_1, F_2, ..., F_n, G_0$ を $\{\text{“}\varepsilon\text{”}\}$ で初期化
SeqBDD $G_1, G_2, ..., G_n$ を $\emptyset$ で初期化
for $i = 1, ..., n$ **do**
  for $j = 1, ..., i$ **do**
    $G_{i-j+1} \leftarrow G_{i-j+1} \cup (\{\text{“}i\text{”}\} * G_{i-j})$
  end for
  for $k = 0, ..., i-2$ **do**
    $F_i \leftarrow F_i \cup (G_k * \{\text{“}S\text{”}\} * F_{i-k-1})$
  end for
end for
return F_n

Algorithm 2 $\mathcal{P}_{n,r}$ を表現する SeqBDD 構築

Input: $n, r \in \mathbb{N}$
Output: $\mathcal{P}_{n,r}$ を表現する SeqBDD F_n

SeqBDD $F_1, F_2, ..., F_n, G_0$ を $\{\text{“}\varepsilon\text{”}\}$ で初期化
SeqBDD $G_1, G_2, ..., G_{r-1}$ を $\emptyset$ で初期化
for $i = 1, ..., n$ **do**
  for $j = \max(1, i+2-r), ..., i$ **do**
    $G_{i-j+1} \leftarrow G_{i-j+1} \cup (\{\text{“}i\text{”}\} * G_{i-j})$
  end for
  for $k = 0, ..., \min(r-1, i-2)$ **do**
    $F_i \leftarrow F_i \cup (G_k * \{\text{“}S\text{”}\} * F_{i-k-1})$
  end for
end for
return F_n

3.3.2 制約を考慮した操作列集合

集合分割の族に対する最適化問題などの実応用を考えると，任意の集合分割を解空間として考える場合は少なく，何らかの制約を満たした集合分割の族を考える場合が多い．このことから制約を考慮した集合分割の族を扱うことは重要であると考えられる．

前節までで扱った制約を考慮しない集合分割の族に対応した操作列 $\mathcal{L}_n$ の生成法を少し変更することでいくつかの制約を満たした集合分割の族に対応する操作列の生成法を得ることができる．本節では 2 通りの制約，各セルの要素数に関する制約およびセルの総数に関する制約について述べる．

まず各セルの要素数に関する制約について述べる．本節では各セルの最大要素数が制限された集合分割のみを含む集合分割の族を考える．例えば 3 要素の集合分割の族 $\{(ab|c), (ac|b), (a|bc), (abc)\}$ は，各セルの最大要素数が 2 以下であるような集合分割のみを含んだものとなっている．このように，各セルの最大要素数を $r\,(r > 0)$ 以下に制限する制約を考えることができる．以上の制約を満たす n 要素の集合分割の族に対応した操作列集合を $\mathcal{P}_{n,r}$ として，$\mathcal{L}_n$ のような生成式を考える．ここで 3.2 節で考えた部分操作列の挙動を再度確認すると，最も大きい要素とそれ以外の要素 $k\,(0 \leq k < n-1)$ 個を含むセルを生成する操作となっている．要素数が r を超えたセルが生成されるのは，この部分操作列において $k \geq r$ となる場合のみであるので，k の上限を $n-1$ 未満ではなく $\min(r, n-1)$ 未満とすればよい．よって，$\mathcal{P}_{n,r}$ は

$$\mathcal{P}_{n,r} = \{\text{“}\varepsilon\text{”}\} \cup (\bigcup_{k=0}^{\min(r-1,n-2)} (\mathrm{Comb}(n-1,k) * \{\text{“}S\text{”}\} * \mathcal{P}_{n-k-1,r}))$$

のように書くことができる．$\mathcal{P}_{n,r}$ を表現する SeqBDD 構築のアルゴリズムについても少しの変更をするのみでよく，以下のように書くことができる．

次にセルの総数に関する制約について述べる．本節ではセルの総数が制限された集合分割のみを含む集合分割の族を考える．例えば 3 要素の集合分割の族 $\{(ab|c), (ac|b), (a|bc), (abc)\}$ は，セルの総数が 2 以下であるような集合分割のみを含んだものとなっている．このように，セルの総数を $t\,(t > 0)$ 個以下に制限する制約を考えることができる．ここで，先述の場合と同様に 3.2 節で考えた部分操作列の挙動を確認すると，任意の $n-k-1$ 要素の集合分割にセルを一つ増やす操作となっていることが分かる．すなわち，セルの総数がちょうど s となる n 要素の集合分割の族と対応した操作列集合を $\mathcal{R}_{n,s}$ ($\mathcal{R}_{i,1} = \{\text{“}\varepsilon\text{”}\}$) とすれば

$$\mathcal{R}_{n,s} = \bigcup_{k=1}^{n-2} (\mathrm{Comb}(n-1,k) * \{\text{“}S\text{”}\} * \mathcal{R}_{n-k-1,s-1})$$

と書くことができる．さらに，セルの総数が t 以下となる n 要素の集合分割の族に対応した操作列集合を $\mathcal{Q}_{n,t}$ とすると

$$\mathcal{Q}_{n,t} = \bigcup_{s=1}^{t} \mathcal{R}_{n,s}$$

であるので，これらによって求める操作列集合が得られる．また，$\mathcal{Q}_n$ を表現する SeqBDD 構築は Algorithm 3 のように書ける．なお，$\mathcal{R}_{n,s}$ を用いることでセルの総数をちょうど t とする制約，t_s 以上 t_t 以下とする場合の制約なども簡単に求めることができる．

ここまでで考えた操作列集合を組み合わせることで別の制約を満たす操作列集合を生成することも可能である．例えば，各セルの要素数が r を超える，各セルの要素数が r 以下かつセルの総数が t 以下となるというような制約はそれぞれ $\mathcal{L}_n \setminus \mathcal{P}_{n,r}$，$\mathcal{P}_{n,r} \cap \mathcal{Q}_{n,t}$ と表現できる．これらについても，$\mathcal{L}_n$, $\mathcal{P}_n$, $\mathcal{Q}_n$ に対応した SeqBDD 構築ができていれば，差集合演算および共通集合演算を SeqBDD において実行するだけで得られる．

Algorithm 3 $\mathcal{Q}_{n,t}$ を表現する SeqBDD 構築

Input: $n, t \in \mathbb{N}$
Output: $\mathcal{Q}_n$ を表現する SeqBDD F_n

SeqBDD $F_{1,1}, F_{2,1}, ..., F_{n,1}, G_0$ を $\{\text{“}\varepsilon\text{”}\}$ で初期化
SeqBDD $F_{1,2}, F_{1,3}, ..., F_{1,t}, F_{2,2}, ..., F_{n,t}$ を $\emptyset$ で初期化
SeqBDD $G_1, G_2, ..., G_n$ を $\emptyset$ で初期化
for $i = 1, ..., n$ **do**
  for $j = 1, ..., i$ **do**
    $G_{i-j+1} \leftarrow G_{i-j+1} \cup (\{\text{“}i\text{”}\} * G_{i-j})$
  end for
  for $s = 1, ..., t-1$ **do**
    for $k = 0, ..., i-2$ **do**
      $F_{i,s+1} \leftarrow F_{i,s+1} \cup (G_k * \{\text{“}S\text{”}\} * F_{i-k-1,s})$
    end for
    if $i = n$ **then**
      $F_n \leftarrow F_n \cup F_{i,s}$
    end if
  end for
end for
return F_n

4. 計算機実験

提案手法による SeqBDD について，メモリ効率・時間効率・圧縮効率を確かめるため，$\mathcal{L}_n$, $\mathcal{P}_{n,r}$, $\mathcal{Q}_{n,t}$ をそれぞれ表現する SeqBDD 構築の実験を行った．上記について，SeqBDD の節点数・SeqBDD の構築時間・SeqBDD に含まれる集合分割の総数 (操作列の総数) を調べることで評価を行う．なお，含まれる集合分割の総数は構築した SeqBDD の根節点から終端節点までを辿ることで求めることができるので，SeqBDD の節点数に比例した時間で算出することが可能である．実験環境には，64-bit Ubuntu 16.04 LTS, Intel Core i7-3930K 3.2GHz CPU，64GB RAM を用いた．

表 1: $\mathcal{L}_n$ を表現する SeqBDD

n	節点数	構築時間 (sec)	含まれる集合分割の総数
10	53	0.000	$\approx 1.15e+5$
20	208	0.003	$\approx 5.17e+13$
30	463	0.003	$\approx 8.46e+23$
40	818	0.017	$\approx 1.57e+35$
50	1,273	0.022	$\approx 1.85e+47$
60	1,828	0.032	$\approx 9.76e+59$
70	2,483	0.051	$\approx 1.80e+73$
80	3,238	0.071	$\approx 9.91e+86$
90	4,093	0.103	$\approx 1.41e+101$
100	5,048	0.137	$\approx 4.75e+115$

表 2: $\mathcal{P}_{n,r}$ を表現する SeqBDD

n	r	節点数	構築時間 (sec)	含まれる集合分割の総数
10	5	95	0.000	$\approx 1.12e+5$
20	5	565	0.000	$\approx 4.19e+13$
30	5	1,435	0.002	$\approx 5.09e+23$
40	5	2,705	0.006	$\approx 6.36e+34$
50	5	4,375	0.009	$\approx 4.63e+46$
60	5	6,445	0.011	$\approx 1.40e+59$
70	5	8,915	0.006	$\approx 1.39e+72$
80	5	11,785	0.016	$\approx 3.87e+85$
90	5	15,055	0.018	$\approx 2.65e+99$
100	5	18,725	0.023	$\approx 4.07e+113$

表 3: $\mathcal{Q}_{n,t}$ を表現する SeqBDD

n	t	節点数	構築時間 (sec)	含まれる集合分割の総数
10	5	128	0.001	$\approx 8.64e+4$
20	5	583	0.006	$\approx 7.95e+11$
30	5	1,338	0.020	$\approx 7.76e+18$
40	5	2,393	0.033	$\approx 7.57e+25$
50	5	3,748	0.059	$\approx 7.40e+32$
60	5	5,403	0.099	$\approx 7.22e+39$
70	5	7,358	0.154	$\approx 7.05e+46$
80	5	9,613	0.243	$\approx 6.89e+53$
90	5	12,168	0.399	$\approx 6.73e+60$
100	5	15,023	0.565	$\approx 6.57e+67$

$\mathcal{L}_n$ を表現する SeqBDD について，含まれる集合分割の総数に対して節点数・構築時間ともに非常に小さくなっていることが確認できる．また，含まれる集合分割の総数がベル数と一致していることが確認でき，3.2 節で導出した生成式および 3.3 節で導出した SeqBDD 構築法が実験的にも正しいことが示された．

$\mathcal{P}_{n,r}$ を表現する SeqBDD について，$\mathcal{L}_n$ を表現する SeqBDD よりも節点数は大きく，構築時間は少し小さくなっていることが確認できる．節点数については，含まれる集合分割のそれぞれで用いられる $\mathrm{Comb}(n,k)$ が違い，$\mathcal{P}_{n,r}$ の方が節点共有が行われにくいためと考えられる．また，構築時間の違いについては，構築アルゴリズムの反復処理回数の差であると思われる．

$\mathcal{Q}_{n,t}$ を表現する SeqBDD については，他 2 つの場合に比べて構築時間が大きくなっていることが確認できる．構築時間の違いについては，構築アルゴリズムの反復処理回数が他 2 つの場合よりも大きいためであると思われる．

しかしながらどの結果についても，含まれる集合分割の総数に対して節点数・構築時間が十分に小さくなっていることが確認でき，提案手法による集合分割の族の表現法は有効であるといえる．

5. まとめ

本稿では集合分割と一対一に対応する操作列を考え，その集合を SeqBDD で表現することで集合分割の族を扱う手法を提案した．また，制約を考慮した場合およびいくつかの制約を考慮した場合の操作列集合の生成法を考え，SeqBDD 構築法とその実験結果を示した．実験結果によって，要素数 100 の場合の膨大な数の集合分割を含む集合分割の族に対しても提案手法による表現において高速かつ少ない節点数で SeqBDD を構築可能であることを確認した．

今後の課題として，まず操作列集合を表現する SeqBDD における演算の考案が挙げられる．SeqBDD において文字列集合特有の演算を考えられたように，集合分割の族においても特有の演算を考えることが可能であると思われる．このような演算によって，有用な制約を満たした集合分割の族を取り出せる可能性があり，その考案が望まれる．さらに，提案手法を基とした操作列集合の ZDD 表現の提案も課題として挙げられる．SeqBDD は文字列表現のため，ZDD における変数順序の制約を緩和したことで実行が困難となった演算がいくつか存在する．提案手法を基とした操作列集合に対して，このような演算を適用することで有用となる制約・演算が得られる可能性があり，ZDD を用いた操作列集合表現は有効となり得ると考えている．また，提案手法を用いた実応用先の検討および実験も視野に入れている．

謝辞

本研究の一部は科研費基盤 (S)15H05711 の助成による．

参考文献

[1] 根本俊男, 堀田敬介. 公平な小選挙区制のための数理モデル. システム/制御/情報 : システム制御情報学会誌, Vol. 49, No. 3, pp. 78–84, 2005.
[2] 茨木俊秀. 組合せ最適化とスケジューリング問題: 新解法とその動向. 計測と制御, Vol. 34, No. 5, pp. 340–346, 1995.
[3] Burcin Bozkaya, Erhan Erkut, and Gilbert Laporte. A tabu search heuristic and adaptive memory procedure for political districting. *European Journal of Operational Research*, Vol. 144, No. 1, pp. 12–26, 2003.
[4] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, Vol. 35, No. 8, pp. 677–691, 1986.
[5] Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin ichi Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E100-A, No. 9, pp. 1773–1784, 2017.
[6] Shin-ichi Minato. πdd: A new decision diagram for efficient problem solving in permutation space. *Proc. of 14th International Conference on Theory and Applications of Satisfiability Testing*, pp. 90–104, 2011.
[7] E.T. Bell. Exponential polynomials. *Annals of Mathematics*, Vol. 35, No. 2, pp. 258–277, 1934.
[8] Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. *Proc. of 30th ACM/IEEE Design Automation Conf.(DAC 1993)*, pp. 272–277, 1993.
[9] E. Loekito, J. Bailey, and J. Pei. A binary decision diagram based approach for mining frequent subsequences. *Knowledge and Information Systems*, Vol. 24, No. 2, pp. 235–268, 2010.