B-02

SeqBDD のメモリ使用効率化手法

An Efficient Method to Reduce Memory Usage for SeqBDDs

池田 祐一 †　　山下 茂 †
Yuichi Ikeda　　Shigeru Yamashita

1. はじめに

近年, Web 技術の発展による, アクセス情報の膨大化や DNA 配列などの大規模データから，有用な情報を取り出すデータマイニングの研究が盛んに行われている [1][2]．その中で，大規模データを効率的に記憶，および高速に演算を行う必要性が高まっている．系列二部決定グラフ (Sequence Binary Decision Diagram: SeqBDD) と呼ばれるデータ構造は，大規模論理関数処理を効率的に行うために広く利用されている二分決定グラフ (Binary Decision Diagram: BDD) や，命題論理や組合せ集合を管理するために考案されたゼロサプ レス型 BDD (Zero-suppressed BDD: ZDD) [3] とよばれるデータ構造から派生したものである．SeqBDD は系列集合を扱うために特化して作られており，データマイニング分野に応用され，画期的な有効性が期待されている．

本稿では，SeqBDD 上で接点間の変数順序を考慮することで，グラフサイズを小さくすることにより，データの記憶に必要なメモリ使用量を削減するための手法について提案する．

2. BDD および SeqBDD

2.1. BDD

Binary Decision Diagram(BDD) は，論理関数を二分グラフとして扱うデータ構造である．BDD を用いることで，論理関数を効率的に表現できることが知られている [4].

BDD では，共通の部分グラフの共有や，後述する簡約化規則を用いる．それによって，表現する論理関数，および BDD の構造の一意性を保った上で，単純な二分グラフとして論理関数を表現するよりも，少ない情報量で論理関数を扱う事ができる．

BDD での変数表現の例を以下に示す．表 1 の真理値表で表される 2 変数の AND 演算を単純な二分グラフと BDD で表現する場合を考える．表 1 の真理値で表される論理をそれぞれ，二分グラフとして表したものを図 1，BDD で表したものを図 2 に記す．図 1，図 2 のグラフからわかるように，二分グラフと BDD を比較した際，BDD は冗長な節点，枝を共有，削除することにより，二分グラフよりも小さいサイズで表現できることが分かる．

表 1　A & B 論理の真理値

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

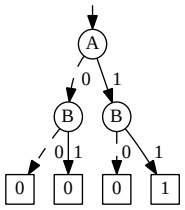

図 1　A & B の論理を表す二分グラフ

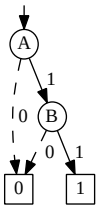

図 2　A & B の論理を表す BDD

2.2. SeqBDD

SeqBDD[5] は，系列集合を効率的に扱うために BDD を拡張したデータ構造であり，系列集合のデータを有向グラフとして扱う．SeqBDD は，図 3 のように，1 文字を円で囲んだ節点，1 を四角で囲った 1- 終端節点，0 を四角で囲った 0-終端節点の 3 種類の節点と，その節点を結ぶ実線の矢印の 1-枝と破線の矢印の 0-枝の 2 種類の枝からなる，閉路のない有向グラフである．SeqBDD での各パスは，1 つの文字列を表す．また，SeqBDD の各節点は文字列集合を持つと考えることができる．その文字列集合は最上位の節点の持つ文字列を 1-枝が指す先の節点の文字列集合の先が指す文字列の先頭に付加したものと，0-枝側の文字列集合との和集合となる．なお，1-終端節点は空の文字列集合を表し，0-終端節点は空集合を表す．

例として，図 3 の SeqBDD は，文字列 {AAG, AG, AU} を表す．この時，図 3 の太線の矢印がたどっているパスでは，文字列”AAG” を表す．

なお，これ以降の図では図の表記を簡約化するため，明示的に 0-終端節点を表記する必要がない図に関して，空集合を表す 0-終端節点及びそれにつながる枝を省略する．

2.3. 簡約化

前節で述べたように，BDD や SeqBDD では簡約化規則を適応することによってグラフのサイズを縮小することができる．本節では，BDD と SeqBDD の簡約化規則，およびそれぞれの

† 立命館大学大学院情報理工学研究科, Graduate School of Information Science and Engineering, Ritsumeikan University

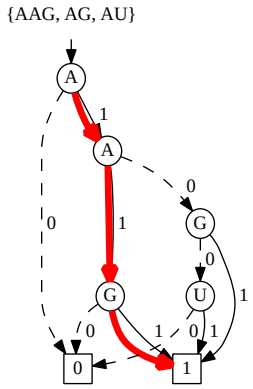

図 3　文字列 {AAG, AG, AU} を表す SeqBDD

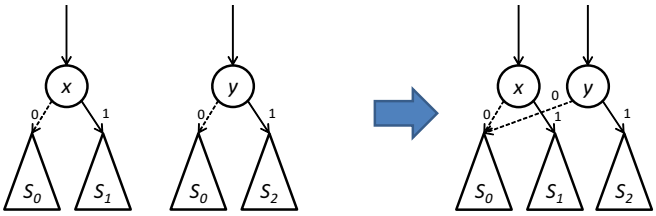

図 4　BDD の節点共有

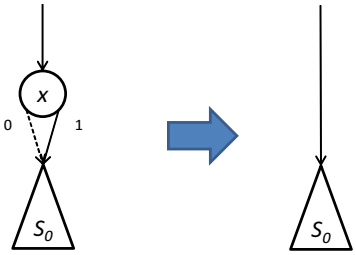

図 5　BDD の簡約化規則

簡約化規則の適応条件が大きく異なる点について示す．

2.3.1　BDD の簡約化規則

BDD では，それ以降の枝，および節点が同じ，すなわちその後に続く部分グラフが共通の節点を共有する．また，ある節点から伸びる 0-枝と 1-枝が共通の節点を指す場合，その始点となっている節点を省略することができる．

それぞれの例を図 4，5 に記す．図 4 では，節点の共有の例を示す．図 4 の x と y はそれぞれ任意の節点を表し S_i は任意の BDD を表している．図 4 の節点 x と節点 y ではそれぞれで同じ BDD “S_0” を指している．この場合，同じ BDD を共有することでグラフ全体のサイズを減らすことができる．共有すると，図 5 の右側の BDD のように表記することができる．

図 5 では，節点の省略の例を示している．図 5 の節点 x は 0-枝と 1-枝で同じ BDD を指している．そのため，この場合の節点 x の変数は考慮する必要がなくなるため，この節点を省略することができる．結果，図 5 の右側の BDD に簡約化することができる．

2.3.2　SeqBDD の簡約化規則

SeqBDD においても，BDD と同様に定められた簡約化規則を用いることで，保持する情報は同一のまま，節点，枝を省略することができる．節点の共有は，BDD と同様に同じ SeqBDD が存在する場合，その箇所を共有することができる．しかし，節点の省略の場合，BDD と SeqBDD では異なる簡約化規則が適応される．

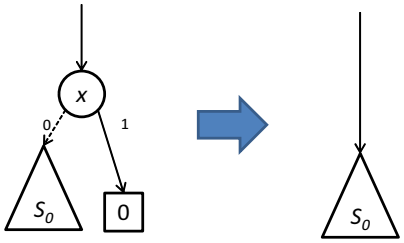

図 6　SeqBDD の簡約化規則

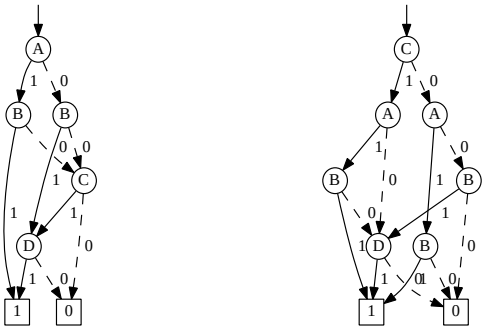

図 7　同じ論理関数を表すサイズの異なる BDD

SeqBDD において節点を省略する場合を考える．対象となる SeqBDD 上に保持する文字列集合に必要のない情報が含まれる節点が，構築，簡約化の際に発生した場合，その節点を省略する．

具体的な例を図 6 に示す．図 6 の左側の SeqBDD のように，任意の節点において，1-枝側に 0-終端節点を示している状態の時，図 6 の右側のようにその節点を省略することができる．

これらの簡約化規則を用いることで，同一の情報を保持したまま，全体のグラフサイズを縮小することができる．

2.4. 変数順序による節点数の変化

BDD および SeqBDD を始めとした BDD を拡張したデータ構造については，その節点の変数の優先順序によって，グラフのサイズが大きく変わることが知られている．図 7 に具体的な例を示す．図 7 の 2 つの BDD はどちらも同じ論理関数 $AB + BD + CD$ を表す BDD である．ただし，左側の BDD では節点の変数順序が $A > B > C > D$ という順序になっているのに対して，右側の BDD では節点の変数順序が $C > A > B > D$ という順序になっている．この例のように変数順序を変化させることで同じ論理を表す BDD にでもそのサイズが異なる可能性がある．

SeqBDD でも処理する変数順序を変えることで同じ系列集合を持つ SeqBDD でも，同様に SeqBDD 全体のサイズが大きく異なる可能性がある．

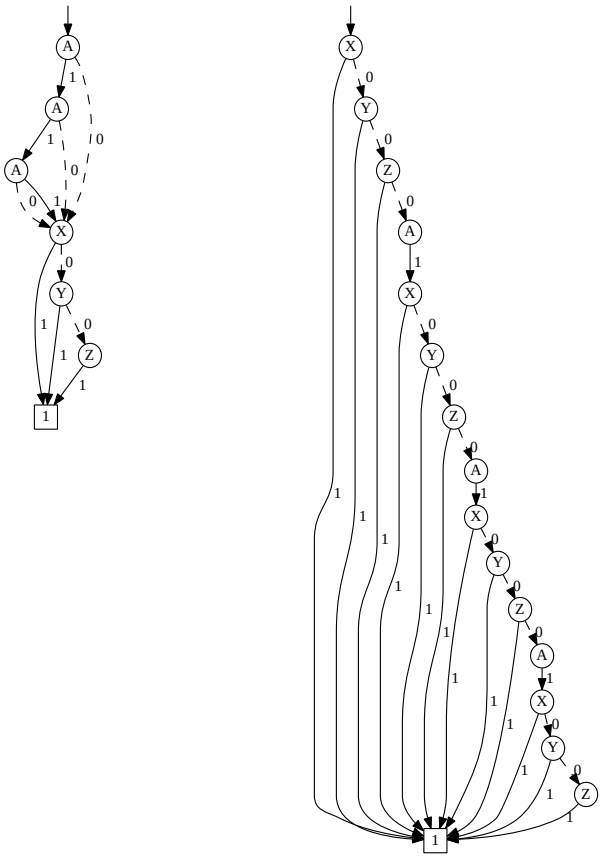

図 8　同じ集合を持つサイズの異なる SeqBDD

図 8 に具体的な例を示す．図 8 の 2 つの SeqBDD はどちらも同じ文字列集合:{X, Y, Z, AX, AAX, AAAX, AY, AAY, AAAY, AZ, AAZ, AAAZ} を表す SeqBDD である．ただし，左側の SeqBDD では節点の変数順序が $A > X > Y > Z$ という順序になっているのに対して，右側の SeqBDD では節点の変数順序が $X > Y > Z > A$ という順序になっている．

この時，左側の SeqBDD の節点数は 1-終端節点を含め 7 節点だが，右側の SeqBDD の節点数は 1-終端節点を含め 16 節点と，倍以上の節点数になる．この変数順によってグラフサイズが大きく変わる特性は，BDD 全般が持ちうる特性であり，他の BDD，および BDD から拡張された他の DD でも有する可能性がある．SeqBDD では節点数が増えるほど，その SeqBDD を保持するために必要な記憶装置の容量も大きくなる，またグラフとしてデータを処理するため，グラフの最長パスが伸びるほど最大処理時間が伸びる．そのため，SeqBDD の節点数およびグラフの最長パスは，基本的に小さいほど好ましい．この特性は，変数順序が変わることで，前述の節点の共有，簡約化が行われる箇所が変わるために存在する．

BDD において，変数順序を入れ替えた際の例を記す．図 9，10 は，どちらも同じ論理関数を表す BDD であり，変数順序を x_0と x_1 で入れ替えている．図 9 の BDD において，節点の省略が行われるのは，S_0と S_1 が共通である場合と，S_2と S_3 が

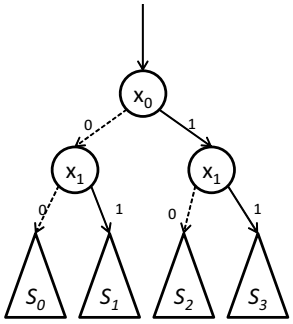

図 9　変数順序が $x_0 > x_1$ である BDD

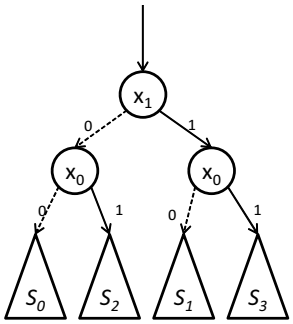

図 10　変数順序が $x_1 > x_0$ である BDD

共通である場合である．一方，同じ論理を表す変数順序の違う図 10 の BDD で節点の省略が行われるのは，S_0と S_2 が共通である場合と，S_1と S_3 が共通である場合である．このように同じ論理を表す BDD でも，変数順序によって簡約化される箇所が異なる可能性があり，そのため変数順序順によって，BDD のサイズは大きく変化する．

BDD 全般の変数順序付け問題の最適解は NP 困難であることが知られており [6]，ある程度の大きさを超えた SeqBDD の変数順序付け問題の最適解を見つけることは非常に困難であると思われる．しかしながら BDD やいくつかの BDD を元に拡張したデータ構造においては，ヒューリスティックな手法がいくつか考案されている [1][7][8].

しかし，BDD の既存手法をそのまま SeqBDD に適応することは困難である．それは BDD と SeqBDD の簡約化が行われる条件が大きく異なっているためである．

その違いを確認するために，SeqBDD において図 9，10 と同じように変数順を入れ替えた例を図 11，12 に示す．図 11，12 に示される SeqBDD は同じ情報を保持する変数順序が異なる SeqBDD である．この時，BDD では変数順を入れ替えてもグラフの最長パスは変化しなかったが，SeqBDD ではグラフの最長パスが変化している事が，図 11，12 から確認できる．

この違いは，BDD が論理関数を評価するため，変数の評価順序は考慮しないことに対して，SeqBDD では系列集合を扱うためである．SeqBDD で単純に変数の評価順序を入れ替えてしまうと，SeqBDD が持つ情報そのものが変わってしまう．例えば，図 9，10 の BDD を SeqBDD として扱った場合を考える．図 9 を SeqBDD として考えたとき，S_3 につながっ

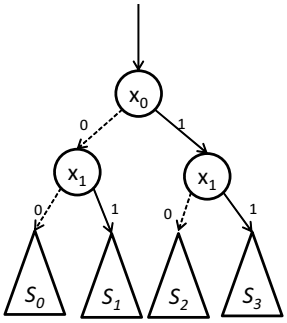

図 11　変数順序が $x_0 > x_1$ である SeqBDD

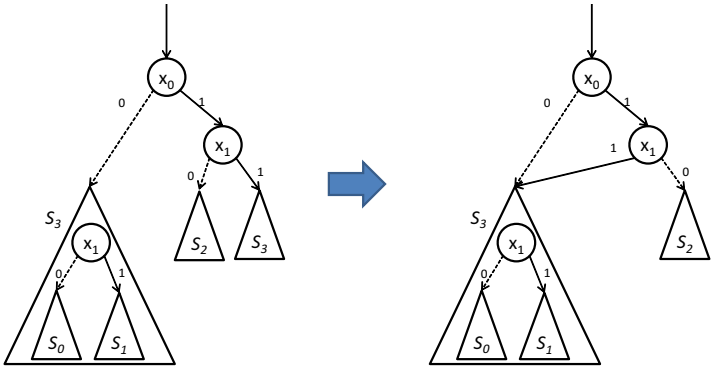

図 13　共通部分を持つ SeqBDD

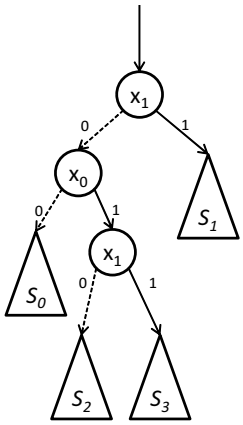

図 12　変数順序が $x_1 > x_0$ である SeqBDD

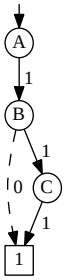

図 14　共通する接頭辞を持つ SeqBDD

ているパスが持つ情報は $x_0x_1\{S_3\}$ を表しているが，図 10 を SeqBDD として考えたとき，S_3 につながっているパスが持つ情報は $x_1x_0\{S_3\}$ を表わすこととなり，2 つの SeqBDD で保持している系列集合が異なることになる．

そのため，同じ情報を保持させたまま，SeqBDD の変数順序を入れ替える場合には，図 11，12 のようにグラフ全体の構造を変える必要がある．また，簡約化が適応される箇所も BDD の場合と違い，変化しない．図 11 の SeqBDD では S_1, あるいは S_3 の箇所が 0-終端節点だった時，その箇所を指している枝の始点である節点を簡約化規則に基づいて，省略することができる．図 12 の SeqBDD でも，S_1, あるいは S_3 の箇所が 0-終端節点だった場合，簡約化規則に基づいて，省略される．このことから分かるように，SeqBDD において，変数順序を変更しても BDD のように簡約化規則が適応される箇所は変化しない．そのため，SeqBDD において全体のサイズを削減するためには，共通の部分グラフが多く存在するようにする必要がある．

SeqBDD において，節点が共有される場合を考える．図 11 の SeqBDD の右側の x_1 以下のグラフが S_3 と同一である場合の例を図 13 に示す．

この時，節点の共有を適応することで，S_3 の箇所を表現する SeqBDD を 1 箇所に集約でき，全体の節点数を減らすことができる．しかし，同様の条件を変数順序を入れ替えた図 12 の SeqBDD に適応しようとした場合，図 12 の SeqBDD 上では，グラフの構造が変わっているため，S_3 と同様の構造を持つ箇所が存在しなくなっている．そのため，節点の共有が行えず，全体の節点数を減らすことができなくなっている．

このように SeqBDD では，変数順序を変えることでグラフ全体の構造が大きく変わるため，節点の共有が行われる箇所が大きく変わる可能性がある．そのため，BDD の変数順序付けの手法をそのまま SeqBDD の変数順序付けの手法として適応することは難しい．

SeqBDD で節点が共有される場合を考える．はじめに文字列集合 {A, ABC} という SeqBDD を作る場合を考える．この時，集合内の各文字列に共有する接頭辞 “A” に関して，図 14 の SeqBDD のように節点を共有する事ができる．次に文字列集合 {C, ABC} という SeqBDD を作る場合を考える．この時，集合内の各文字列に共有する接尾辞 “C” に関して，図 15 の SeqBDD のように節点を共有する事ができる．最後に文字列集合 {B, ABC} という SeqBDD を作る場合を考える．この時，集合内の共通する文字列は，“B” であるが，この文字列集合で SeqBDD を構築した場合，図 16 のようになり，変数”B” に対応する節点は共有されない．すなわち，SeqBDD で節点の共有を行うためには，できるだけ多くの共通する接頭辞，接尾辞を優先的に使用する必要がある．

本稿では SeqBDD に特化した変数順序付け手法を提案する．

3. 提案手法

本稿で提案する SeqBDD の変数順序付け手法は，前述した SeqBDD の節点共有の特性を利用する．

SeqBDD で節点が共有できるのは，その中に含まれる文字列集合が共通である時である．さらに，その文字列が頻出であ

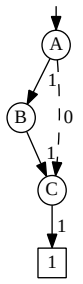

図 15　共通する接尾辞を持つ SeqBDD

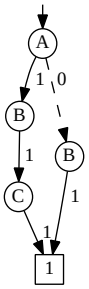

図 16　共通する部分文字列を持つ SeqBDD

るほど共有される．この時，共通である必要があるのは，単一の文字列ではなく，その節点が持つ文字列集合全体である．そのため，文字列の接頭辞，接尾辞が共通であれば，その節点まで，あるいはその節点からの文字列集合は共通である可能性が高い．

上記の条件を考慮した上で，対象の SeqBDD のグラフサイズを削減する変数順序付け手法を本稿では提案する，本手法では，SeqBDD に含まれている文字列集合から，できるだけ多くの共通する接頭辞，接尾辞での文字列を探索し．その文字列の文字順序を採用する．そうすることで対象となる SeqBDD で接点の共有をより多く行い，全体の SeqBDD のグラフサイズの削減を行う．

提案手法のアルゴリズムを，Algorithm1 に示す．対象となる SeqBDD の文字列集合に含まれるすべての接頭辞となる部分文字列と，接尾辞となる部分文字列の出現回数と文字列長を計算する (1-7 行目)．この状態から SeqBDD に含まれる全ての変数について，変数順が求まるまで以下の手順を繰り返す．計算した部分文字列集合から，文字列長と出現回数の積が最大となるものを探す (9 行目)．見つかった文字列の並び順を変数順として保存する．その際にすでに選択した変数順に反する順が含まれる場合，その箇所を無視して変数順とする (10-14 行目)．この手順をすべての変数順が定まるまで繰り返す．

上記の手順を繰り返して，最終的に全ての変数について変数順が求まった時点で，その変数順を対象となる SeqBDD の変数順序として決定する．

4. まとめ，展望

本稿では，SeqBDD の持つ特性を利用した SeqBDD のサイズを削減する変数順序付けの手法を考案した．

Algorithm 1 : 当たられた SeqBDD の変数順を決定する

Require: S: 変数順を決定する SeqBDD に含まれる文字列集合

1: **for all** $str \in S$ **do**
2: $count[str] += 1$
3: **for** $i = 1$ to $i = str.length - 1$ **do**
4: $count[str$ の先頭 i 文字$] += 1$
5: $count[str$ の終端 i 文字$] += 1$
6: **end for**
7: **end for**
8: **while** すべての変数について順序が決まるまで **do**
9: $max\_str = max(count[str] * str.length)$
10: **for** $i = 1$ to $i = max\_str.length - 1$ **do**
11: **if** 決定済の変数順で $max\_str[i] < max\_str[i+1]$ ではない場合 **then**
12: *max_str[i]* と *max_str[i+1]* について，変数順を $max\_str[i] > max\_str[i+1]$ とする
13: **end if**
14: **end for**
15: **end while**

今後の課題として，この手法を適応し，実際に作成される文字列集合に対して，有効な結果が得られるかを検証していく．また，本稿は特定のデータに依存しない手法を提案したが，実際にデータマイニングに用いられるデータごとに異なる特性を持つ可能性もある．そのため，そのデータごとに有効な特性があるかを調査し，その特性に合わせた手法も検討を行っていく．

参考文献

[1] Hand, D. J., Mannila, H. and Smyth, P.: *Principles of data mining*, MIT press (2001).

[2] Han, J., Kamber, M. and Pei, J.: Data Mining: Concepts and Techniques, (The Morgan Kaufmann Series in Data Management Systems) (2006).

[3] Minato, S.-i.: Zero-suppressed BDDs for set manipulation in combinatorial problems, *Design Automation, 1993. 30th Conference on*, IEEE, pp. 272–277 (1993).

[4] Bryant: Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Computers*, Vol. 35, No. 8, pp. 677–691 (online), DOI: 10.1109/TC.1986.1676819 (1986).

[5] Loekito, E., Bailey, J. and Pei, J.: A binary decision diagram based approach for mining frequent subsequences, *Knowledge and Information Systems*, Vol. 24, No. 2, pp. 235–268 (2010).

[6] TANI, S.: The complexity of the optimal variable ordering problems of shared binary decision diagrams, *4th International Symposium on Algorithms and Computation, 1993*, Vol. 762, pp. 389–398 (1993).

[7] Minato, S.-i., Ishiura, N. and Yajima, S.: Shared bi-

nary decision diagram with attributed edges for efficient Boolean function manipulation, *Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE*, IEEE, pp. 52–57 (1990).

[8] 岩崎玄弥， 湊真一ほか: 頻出パターンマイニングのためのゼロサプレス型 BDD の変数順序付け方法とその評価，電子情報通信学会論文誌 D， Vol. 91, No. 3, pp. 608–618 (2008).