Loekito, E., Baily, 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.



理論談話会#3 2016/5/11

吉野 大介

Contents

リサーチクエスチョン

- BDDを用いて系列をコンパクトに表現することはできるか？
- 頻出パターンマイニングにおけるBDD利用のメリットは？
- BDDを用いた頻出パターンマイニングはパターン成長アプローチの最先端となり得るか？

個人的な興味としては，

- BDDやZDDで経路列挙を行う際，リンクの重複を許したり，時間拡大ネットワークに拡張しようとすると，系列を扱わなければ列挙が非効率であり，BDD/ZDDの延長で何とか表現する手法はないか？
- 文字列の頻出パターンを圧縮する手法であるSeqBDDを経路列挙に適用できないか？

パターン成長アプローチ

■頻出パターンマイニングとは

- 2値のデータベースから，その中の多くのレコードで成立している条件または規則をすべて列挙する問題.
- 要は，データベース中に高頻度に存在するある条件パターンを全て列挙する問題のこと.

■パターン成長アプローチとは

- 頻出パターンマイニングを解くための一つのアプローチ.
- Prefixspan (Pei et al., 2004)
- WAP-mine (Pei et al., 2000)
- PLWAP (Ezeife and Lu, 2005)
- これまでに色々と開発されているが，長い系列が存在する場合に計算コストが高くなる点が欠点.

BDD / ZDD

■BDD(Binary Decision Diagram)

- 論理関数のグラフによる表現.
- 論理関数のそれぞれの変数について，0, 1の値を導入した結果を2分岐の枝（0-枝，1-枝）で場合分けし，最終的に得られる論理関数の値を2値の定数節点（0-終端，1-終端）で表現.
- BDDでは場合分けする変数の順序を固定し，「冗長節点の削除（0-枝と1-枝の行き先が同じ接点を削除する）」「等価節点の共有（入力変数ラベルが一緒で0-枝同士・1-枝同士の行き先が同じ接点を共有する）」という2つの縮約条件を適用することで，論理関数をコンパクトかつ一意に表すことが可能.

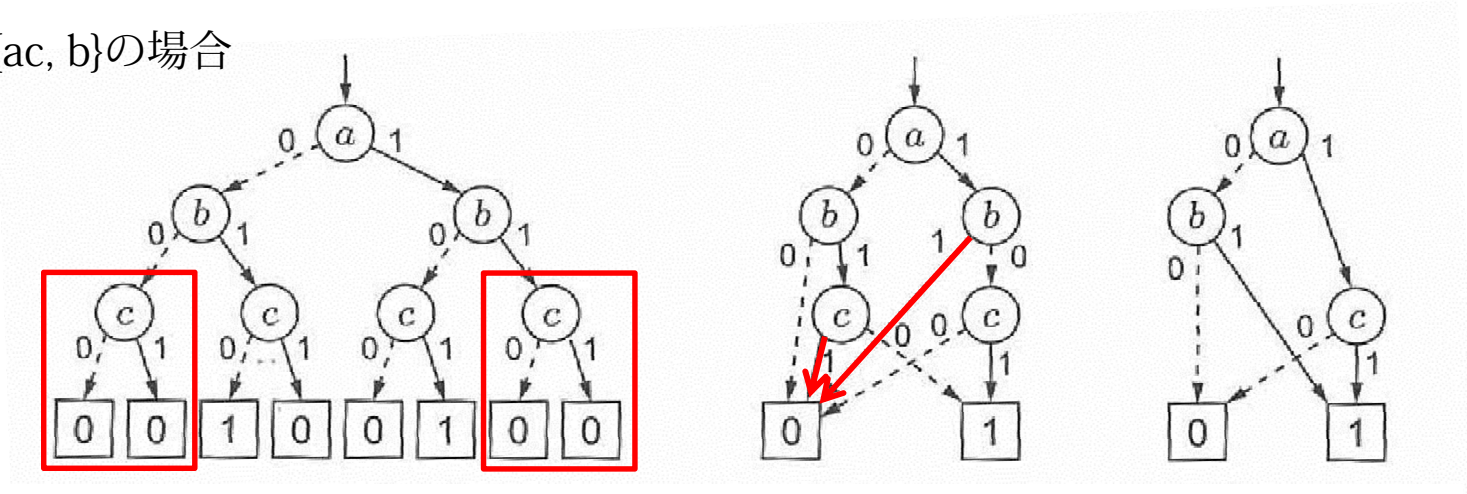
■ZDD(Zero-suppressed BDD)

- ZDDは組合せ集合を表現するために特化したBDDであり，冗長節点の削除の条件がBDDとは異なる.
  - BDD：0-枝と1-枝が同じ節点を指しているときに冗長として取り除く
  - ZDD：上記の節点は取り除かず，1-枝が0-終端を指している場合に取り除く（組み合わせ集合に無関係＝一度も出現しないアイテムに関する節点が自動的に削除されるため，組み合わせが疎な場合に効果が大きい）

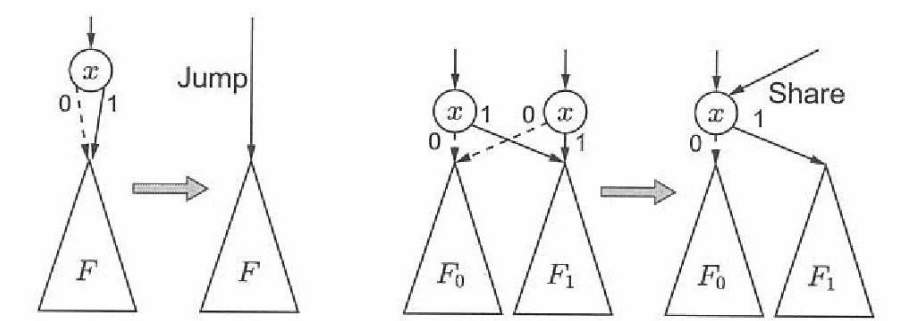
BDD / ZDD

■二分木→BDD→ZDD

組合せ{ac, b}の場合

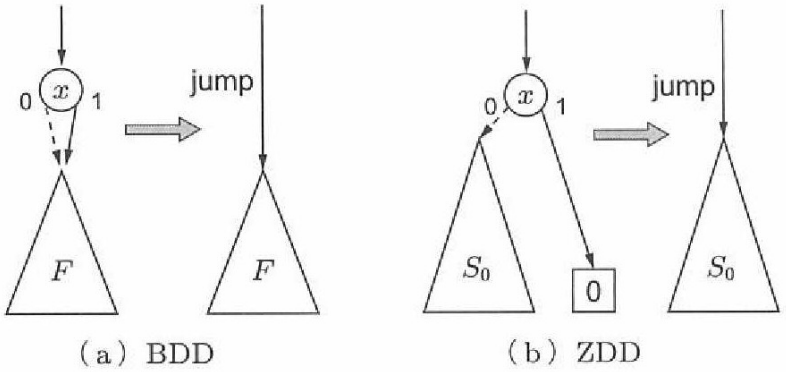


■BDDの簡約化規則



冗長節点の削除　　等価接点の共有

■BDD・ZDDの冗長節点削除規則



（a）BDD　　（b）ZDD

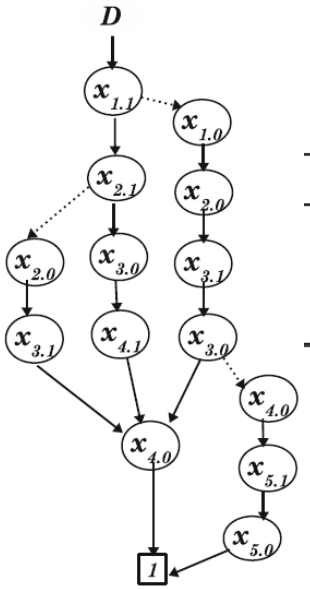
BDD：0/1-枝の行き先が同じ場合読み飛ばす
ZDD：1-枝が0-終端を直接指す場合読み飛ばす

ZDDによる系列の表現

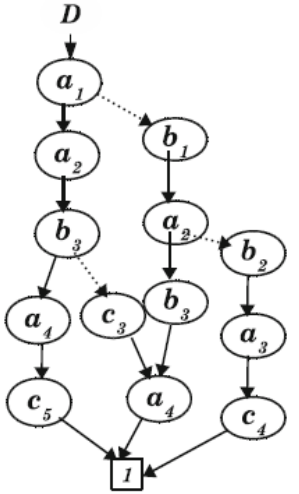
- 通常のZDDは組み合わせ集合に最適化されており，系列集合に最適化されていない（二分木構築の際の変数の順序が固定されているため）．
- Kurai et al. (2007)がアイテムの種類と出現位置の二つを組みにして符号化（ナイーブ型・バイナリ型）することにより系列の表現を試みているが，データ量がコンパクトにならない問題がある．
- 吉野の春大会論文はこの手法に近い．

Symbol	Binary Code (v_1, v_0)	Itemset Encoded
a	$(0,1)$	$x_{j.0}$
b	$(1,0)$	$x_{j.1}$
c	$(1,1)$	$x_{j.1}x_{j.0}$

Sequence	Itemset Encoded
$p_1 = aabac$	$x_{1.0}, x_{2.0}, x_{3.1}, x_{4.0}, x_{5.1}, x_{5.0}$
$p_2 = baba$	$x_{1.1}, x_{2.0}, x_{3.1}, x_{4.0}$
$p_3 = aaca$	$x_{1.0}, x_{2.0}, x_{3.1}, x_{3.0}, x_{4.0}$
$p_4 = bbac$	$x_{1.1}, x_{2.1}, x_{3.0}, x_{4.1}, x_{4.0}$



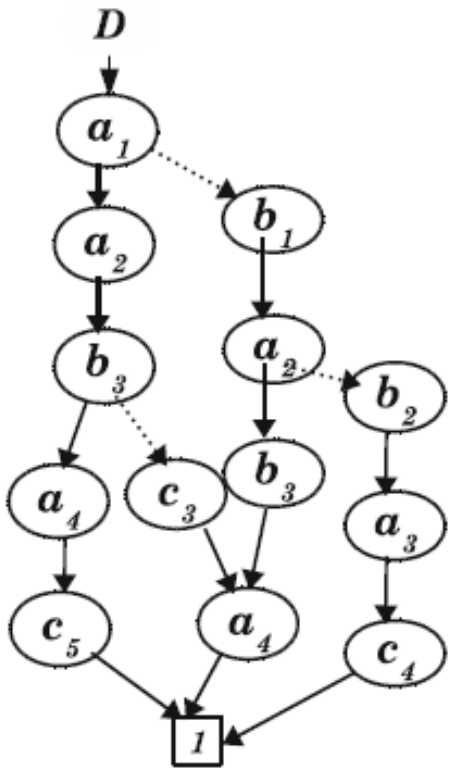
Sequence	Itemset Encoded
$p_1 = aabac$	a_1, a_2, b_3, a_4, c_5
$p_2 = baba$	b_1, a_2, b_3, a_4
$p_3 = aaca$	a_1, a_2, c_3, a_4
$p_4 = bbac$	b_1, b_2, a_3, c_4



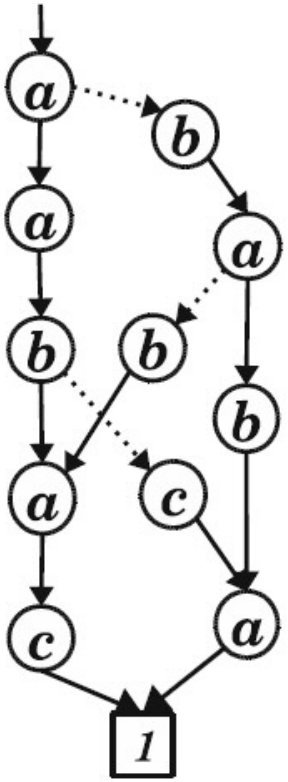
SeqBDD

■SeqBDD(Sequence BDD：系列二分決定グラフ)とは

- 0-枝だけがその親に対して順序づけられていて1-枝については順序付けの制約を外したZDD（0-枝側はZDDと同じ順）.
- ある文字が1つのパスの中に複数回出現することが許される.



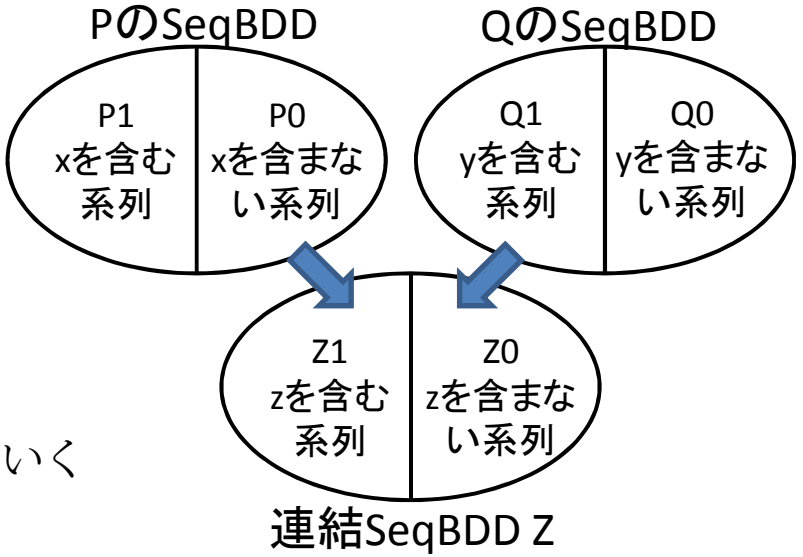
符号化して系列データに対応したZDD



SeqBDD

SeqBDD

■SeqBDD同士の集合演算(Algorithm 1)



インプット：

PとQのSeqBDDs

アウトプット：

PとQの連結SeqBDD Z（$P \cup Q$）

手順：

※以下，PとQに含まれる節点をそれぞれ探索していく

1. Pが0-終端節点の場合，Z=Qを出力する．
2. Qが0-終端節点の場合，Z=Pを出力する．
3. PとQが同一の節点ならば，どちらも等しい文字列集合を示しているのでPを出力する．
4. $Z = node(z, Z_1, Z_0)$, $P = node(x, P_1, P_0)$, $Q = node(y, Q_1, Q_0)$
5. case $x = y$：xはPとQを構成する系列の先頭における（辞書順で）最小の文字
   → xが出力される節点zになる．その1-枝（$Z1$）はxから始まるすべての系列を含み（$Z_1 = P_1 \cup Q_1$），0-枝（Z_0）は残りの系列を含む（$Z_0 = P_0 \cup Q_0$）．
6. case $x < y$：xはPに含まれるすべての系列の先頭よりも小さい
   →xが出力される節点zになる．xから始まるすべての系列P_1にあるので，Z_1=P_1になる．
   残る系列は$P_0 \cup Q$の中に存在するので，Z0= $P_0 \cup Q$ となる．
7. case $x < y$： 6とは逆の条件のため対照的な演算となる．
8. Zを返す．

SeqBDD

■系列データからSeqBDDを構築するアルゴリズム(Algorithm 2)

インプット：

系列データD

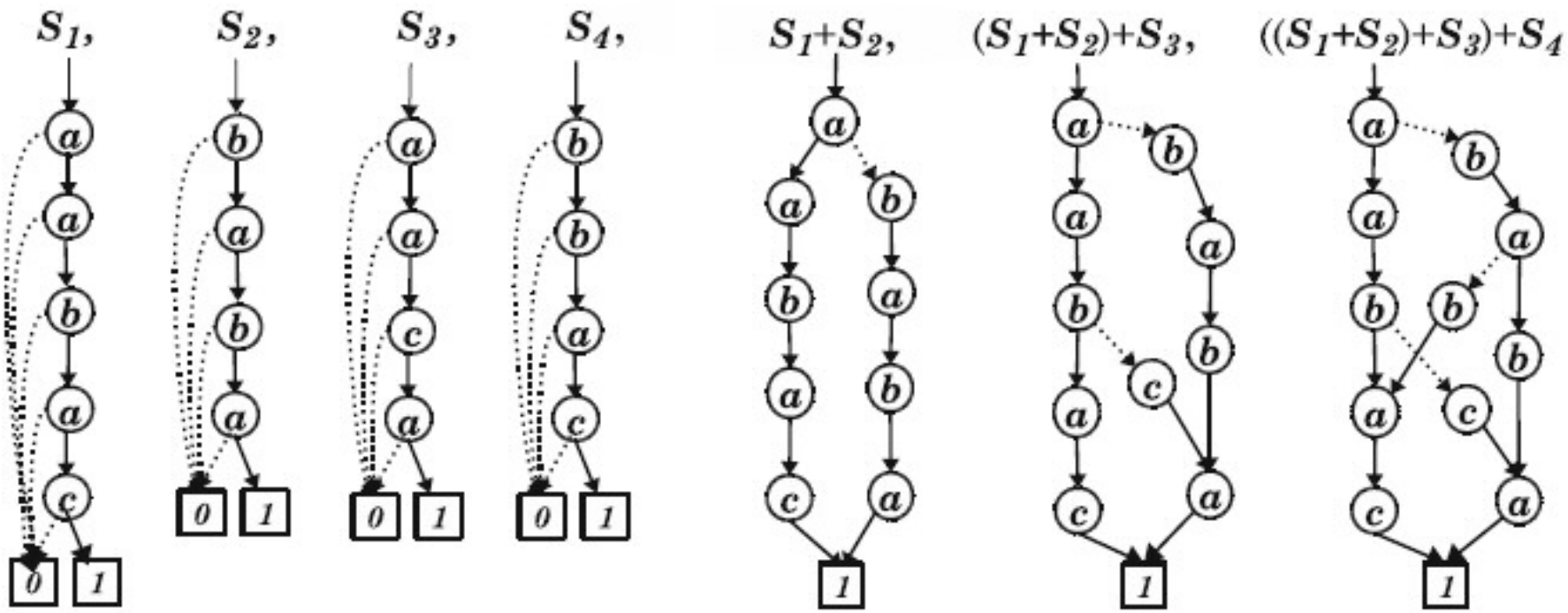
アウトプット：

seqBDD *initDB*

手順：

1. *initDB*=0（SeqBDDを初期化する）
2. **for** each sequence s in D **do**
3. 文字sを読み込み，sをもつSeqBDD P_sを構築する
4. P_sと*initDB*を統合（$initDB \cup P_s$）する（Algorithm 1）.
5. 終了

SeqBDD



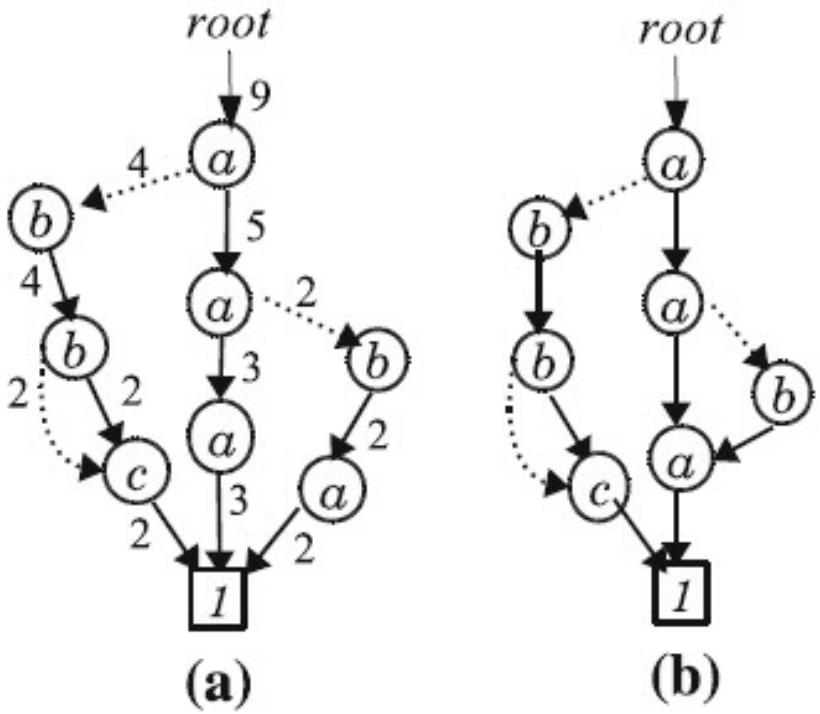
S1～S4の統合SeqBDDを作る場合（Algorithm 1とAlgorithm 2に準拠）

- まずは各系列のSeqBDDを作る（S1～S4）
- 次に，1ペアずつSeqBDDの統合を行う．各ノードを比較し，辞書順に文字が含まれているかどうかをチェック．含まれていれば1枝を，そうでなければ0枝を追加
- 次のノードに移り，同じ処理を行う．
- 終端ノードに到着したら終了，SeqBDDをアウトプット．
- アウトプットとして得られたSeqBddを次のペアと統合し，全部終わったら終わり．

Weighted SeqBDD

■Weighted SeqBDDとは

- SeqBDDの各エッジにウェイトを付与したもの.
- 各エッジのウェイトから各系列の発生頻度を表現することができる.
- 構築アルゴリズムはSeqBDDとほぼ同じ.



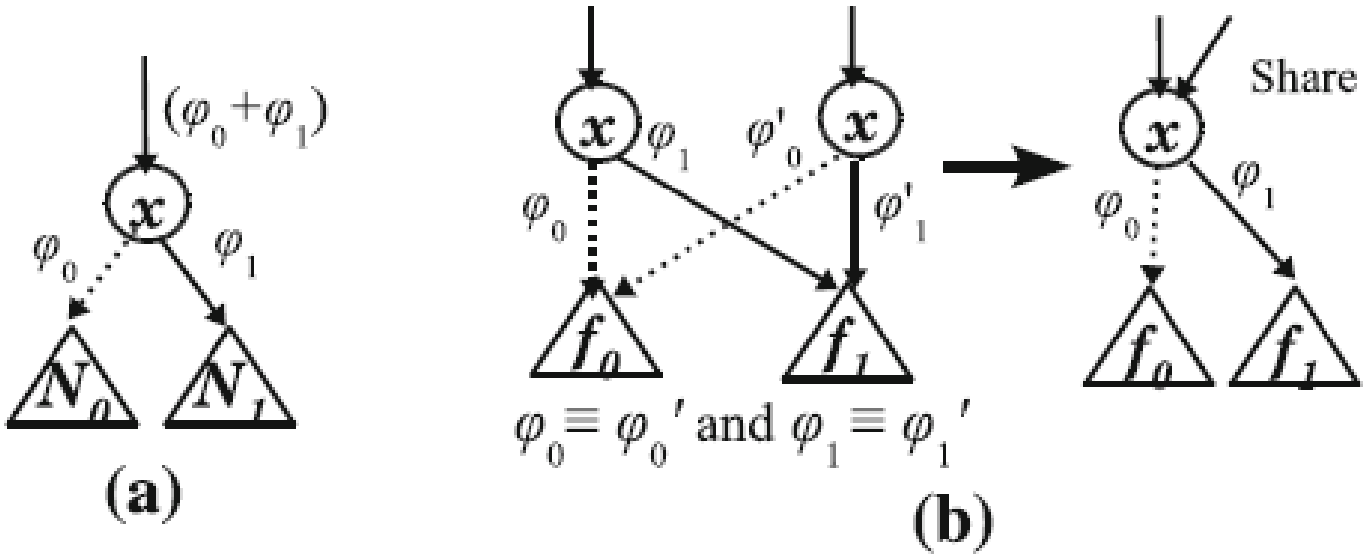
$$S = \{aaa: 3, aba: 2, bc: 2, bbc: 2\}$$

Weighted SeqBDD

■Weighted SeqBDDの構造

SeqBDDとほぼ同じだが，リンクにウェイトがかかるのでノードの共有やSeqBDD同士の統合の際に処理が若干異なる．

- Node $N = \langle \varphi, \vartheta \rangle$; φはノードNに入ってくるリンクのウェイト
- ϑ =node(x, N_1, N_0); ϑはSeqBDDのノード
- $\varphi = \varphi_1 + \varphi_0$　N_1とN_0のウェイトの合計がNのウェイト
- 0-枝と1-枝が同じ節点を指しており，なおかつそれぞれのウェイトが等しい場合にのみ節点が共有化される．



Weighted SeqBDD

■Weighted SeqBDD同士の集合演算

- SeqBDDとほぼ同じだが，ウェイトの計算が途中に入る点が異なる．

Algorithm 1 SeqBDD's **add()** operation, returns the set-union between two sets of sequences

Input: P and Q are the input Weighted SeqBDDs, each of which represents a set of sequences
Output: $\langle Z.weight, Z\rangle$: the Weighted SeqBDD containing set-union between P and Q
Procedure:

1: **case** P **is a sink-**0 **node** : **return** $Z = Q$
2: **case** Q **is a sink-**0 **node** : **return** $Z = P$
3: **case** $Q = P$ **is a sink-**1 **node** :
4: **Calculate weight:** $Z.weight = weight(P) + weight(Q)$
5: **return** $\langle Z.weight, Z\rangle$
6: Let $Z = \mathrm{node}(Z.var, Z_1, Z_0)$, $P = \mathrm{node}(x, P_1, P_0)$, $Q = \mathrm{node}(y, Q_1, Q_0)$
7: **case** $x = y$:
8: $Z.var = x$; $Z_1 = \mathrm{add}(P_1, Q_1)$; $Z_0 = \mathrm{add}(P_0, Q_0)$
9: **case** x **has a higher index than** y :
10: $Z.var = x$; $Z_1 = P_1$; $Z_0 = \mathrm{add}(P_0, Q)$
11: **case** x **has a lower index than** y : **return** $\mathrm{add}(Q,P)$
12: **Calculate weight:** $Z.weight = weight(Z_0) + weight(Z_1)$
13: **return** $\langle Z.weight, Z\rangle$

Mining frequent subsequences

■頻出部分列問題のためのSeqBDDの構築アルゴリズム※(Algorithm 3)

※データベース内で頻出する部分列を列挙し，SeqBDDを構築する問題

インプット：

inputDB：インプットデータから作成したSeqBDD

min_support：頻度に関する閾値

f-list：*inputDB*の発生頻度に関するアイテムリスト

アウトプット：

allFS：頻出部分列を含むSeqBDD

手順：

1. （低頻度データの枝刈り）inputDBを確認し，*min _support*以下の発生頻度の系列を削除する.
2. （終了条件）*inputDB*が空の場合，終了.
3. （終了条件）キャッシュ（後述）に同じアウトプットがある場合，終了.
4. （頻出部分列seqBDDの初期化）*allFS*=1

つづく

Mining frequent subsequences

■頻出部分列問題のためのSeqBDDの構築アルゴリズム(Algorithm 3)

前頁からのつづき

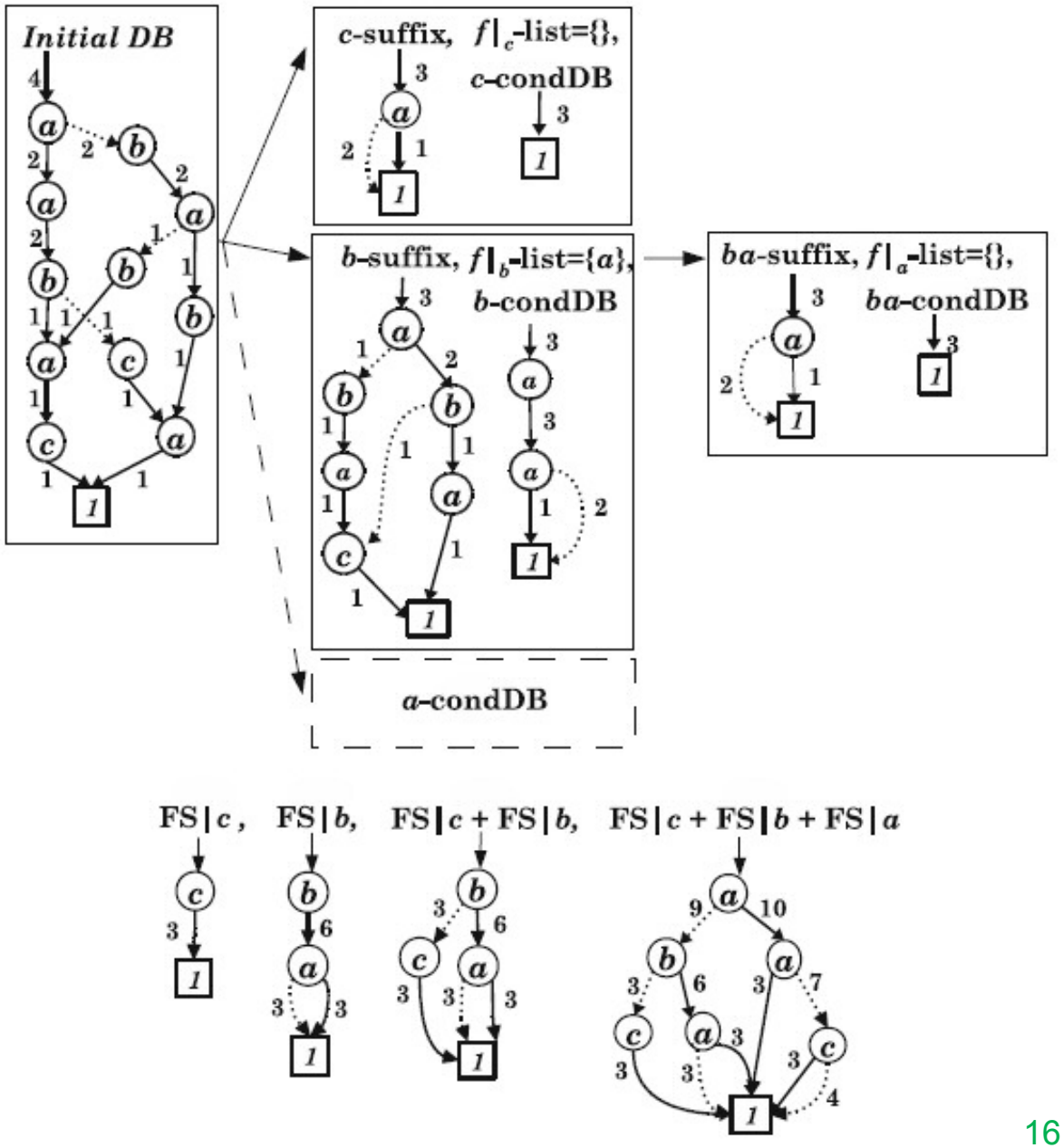
5. **for** each item x in *f-list* **do**
6. （*x-suffix tree*の計算）$suffixTree(inputDB, x)$
   ※xが頭につく系列を列挙
7. （*x-suffix tree*の中に含まれる頻出アイテムを列挙）$f|_x$-*list*
   ※xのあとに続く文字の中で頻出するアイテムを列挙
8. （*x-suffix tree*の低頻度発生アイテムの枝刈り）*x-condDB*
   ※*x-suffix tree*の中で$f|_x$-*list*に含まれないアイテムを削除
9. （*x-condDB*から接頭辞がxとなっている頻出部分列の探索※seqBDD構築）
   x-FS=x×**SeqBDDMiner**(*x-condDB*, *min_support*, $f|_x$-*list*)
10. （seqBDDの更新）*allFS*=*add*(*x-FS*, *allFS*)
11. **end for**
12. （頻出部分列seqBDDをキャッシュとして保存）
    patternCache[*inputDB*] = *allFS*
13. **return** *allFS*

Mining frequent subsequences

S1; p1=aabac
S2; p2=baba
S3; p3=aaca
S4; p4=bbac
*$min\_support$=3とする（出現回数の閾値）

cを含むアイテムに着目
S1, S3, S4
c-suffix
S1; 接尾辞なので{}
S3; a
S4; 接尾辞なので{}
f|c-list={}
※出現回数3回以上の頻出無し
c-condDB={c: 3}→FS|C
※cが含まれる系列は{c}が頻出

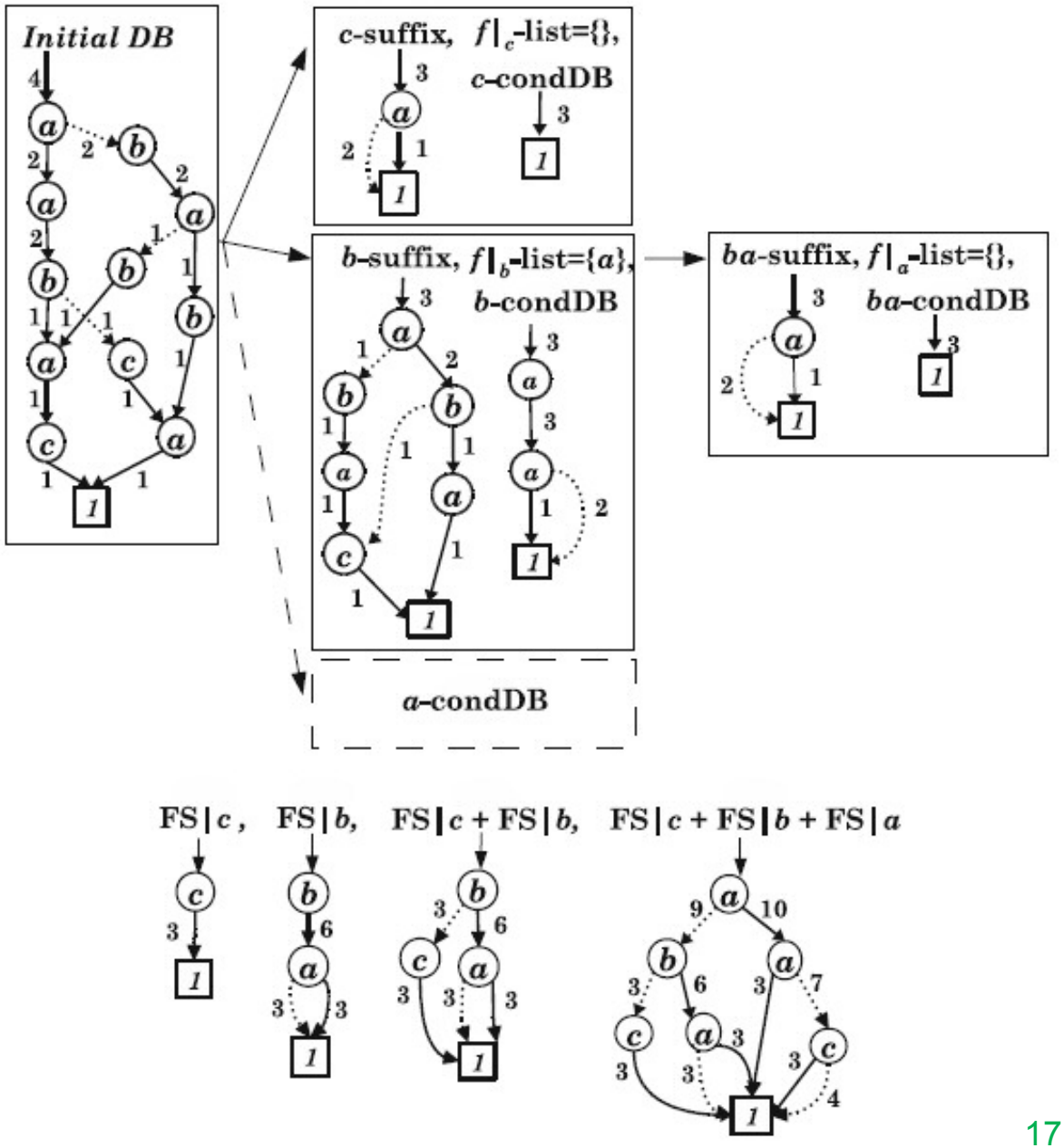
bを含むアイテムに着目
S1, S2, S4
c-suffix
S1; ac
S2; aba
S4; bac
f|c-list=a（aが3回出現）
b-condDB={b: 3, ba3}→FS|b
※bが含まれる系列は{b, ba}が頻出



Mining frequent subsequences

S1; p1=aabac
S2; p2=baba
S3; p3=aaca
S4; p4=bbac

aを含むアイテムに着目
S1, S2, S3, S4
a-suffix
S1; abac
S2; ba
S3; aca
S4; c
f|c-list={a, c}
※aとcが3回以上出現
c-condDB={a: 4, ac:3, aa:3}→FS|a



Performance study

■Dataset

- DNA sequence data sets: 4 letters (A, C, G, T) → yeast.L200, DENV1
- Protein sequence data sets: 20 letters → snake, PSORTb-ccm
- Weblog data sets → gazelle, davinci
- The synthetic data sets → C2.5.S5.N50.D40K

Table 5 Data set characteristics and a proposed categorization based on the average sequence length

Data set name	$\|D\|$	N	L	C	Category
yeast.L200	25	4	129	35	Short
DENV1	9	4	50	50	Long
snake	174	20	25	25	Short
PSORTb-ccm	15	20	50	50	Long
gazelle	29,369	1,451	652	3	V.short
davinci	10,016	1,108	416	2	V.short
C2.5.S5.N50.D40K	17,808	50	42	3	V.short

$|D|$ number of sequences in the data set, N number of items in the domain, L maximum sequence length, C average sequence length, *V.short* very short

Performance study

■SeqBDD's Compactness

- $fanOut = 1 - \frac{|SeqBDTree|}{total\ number\ of\ elements}$ ※二分木でどれだけ圧縮できるか
- $fanIn = \frac{|\quad\quad| \ |SeqBDD|}{|SeqBDTree|}$ ※BDD化することで更にどれだけ圧縮できるか
- $|SeqBDTree|$ =the number of nodes in the SeqBDTree

*SeqBDTree: BDDになる前の二分木の状態

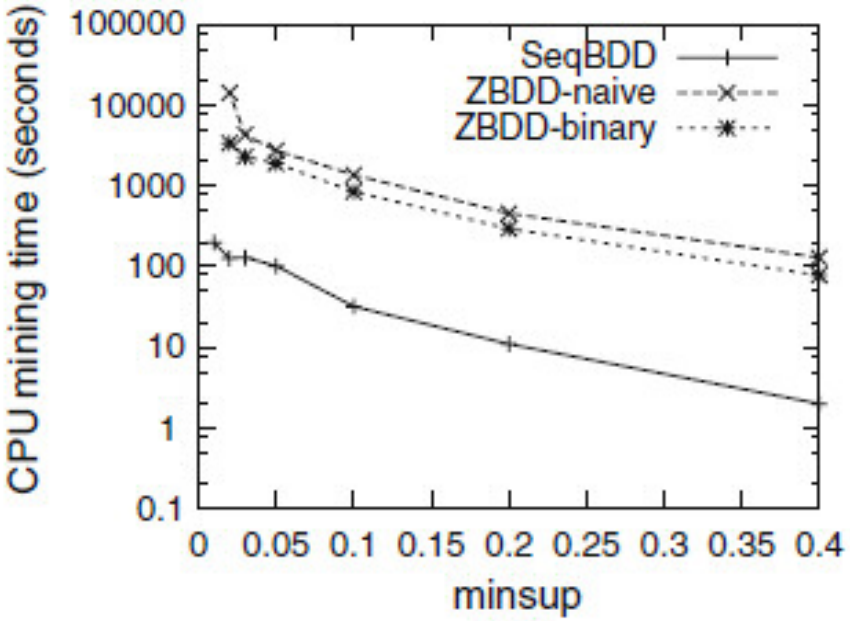
→文字列間の類似性が低いデータは二分木の節点の共有が少なくデータの圧縮効果が小さい

Dataset name	$fanOut$	$fanIn$	Category
yeast.L200	0.22	0.12	Similar
DENV1	0.53	0.07	Similar
snake	0.52	0.10	Highly similar
PSORTb-ccm	0.024	0.004	Dissimilar
gazelle	0.42	0.16	Similar
davinci	0.59	0.03	Similar
C2.5.S5.N50.D40K	0.52	0.18	Highly similar

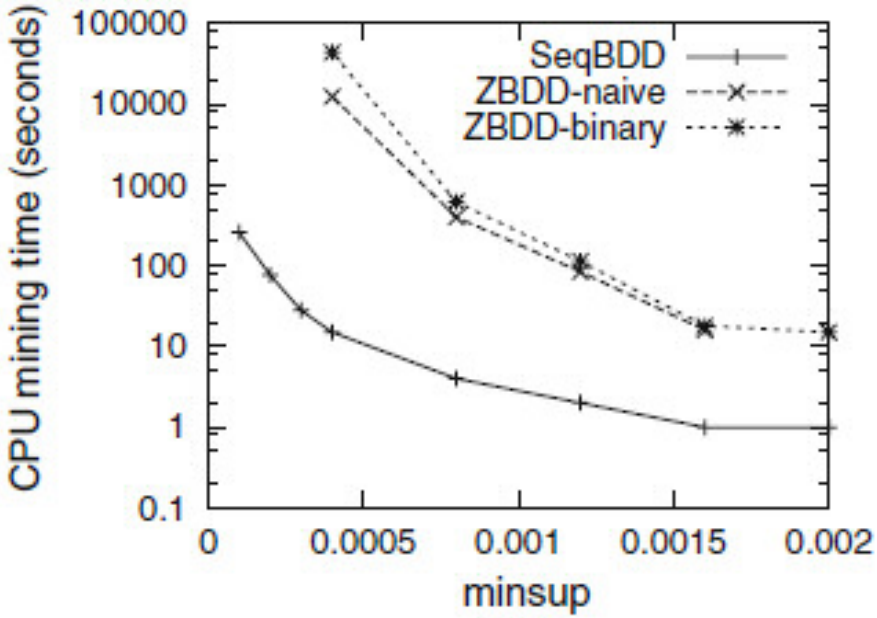
Performance study

■Runtime performance of the mining algorithm

- snake(a) and davinci(b)データセットをもとにseqBDDとZDD（ナイーブ符号化＆バイナリ符号化）による頻出部分列パタンの計算速度を頻度閾値*minsup*を変えつつ算出.
- いずれもSeqBDDはZDDの10倍以上の計算速度.



(a) snake

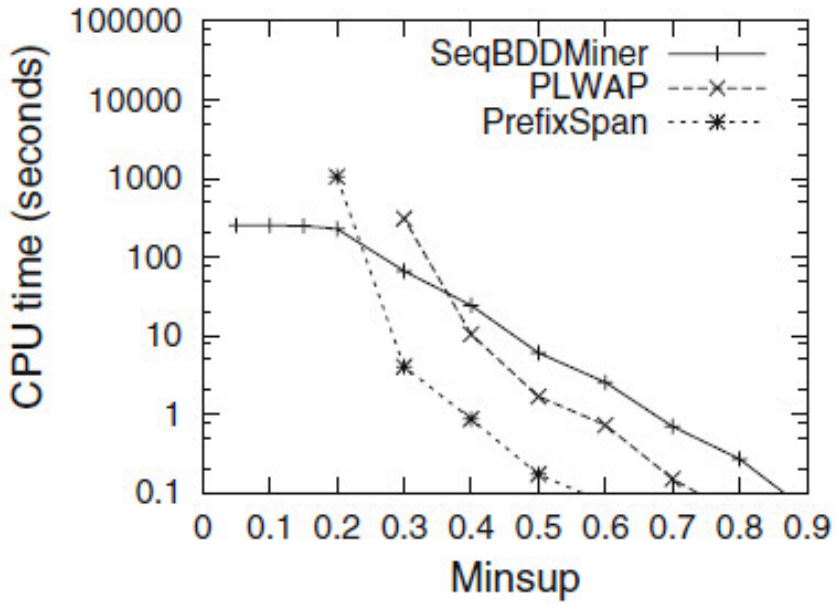


(b) davinci

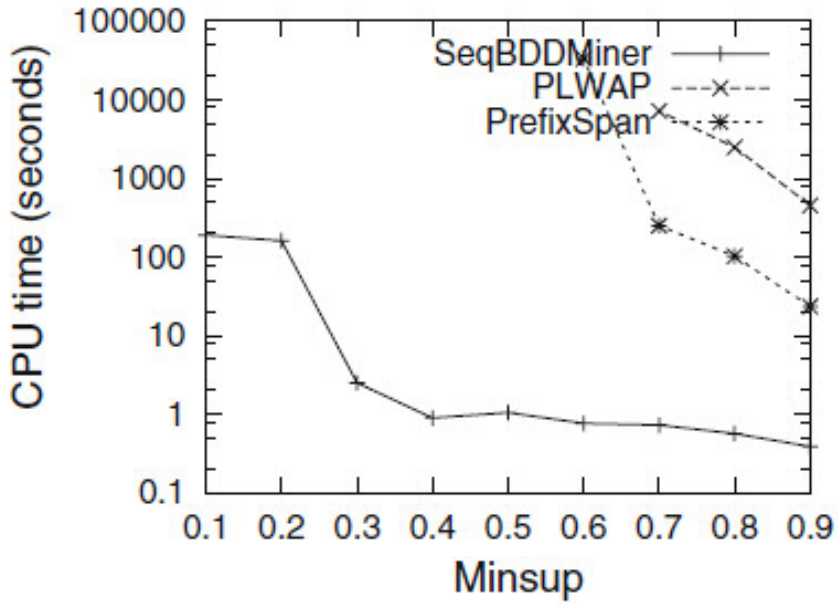
Performance study

■Runtime performance of the mining algorithm

- yeast.L200(a) and DENV1(b)データセットをもとにSeqBDDMinerと既往マイニング手法（PLWAP，PrefixSpan）それぞれの頻出部分系列パタンの計算速度を頻度閾値*minsup*を変えつつ算出.
- yeast.L200は閾値70％ではPLWAPの10倍，PrefixSpanの100倍遅い．しかし，閾値が5％まで下がるとSeqBDDでなければ計算ができない．DENV1も同様.
- SeqBDDのスケーラビリティの優位性（全列挙ならばBDDが優れる）.



(a) Yeast.L200

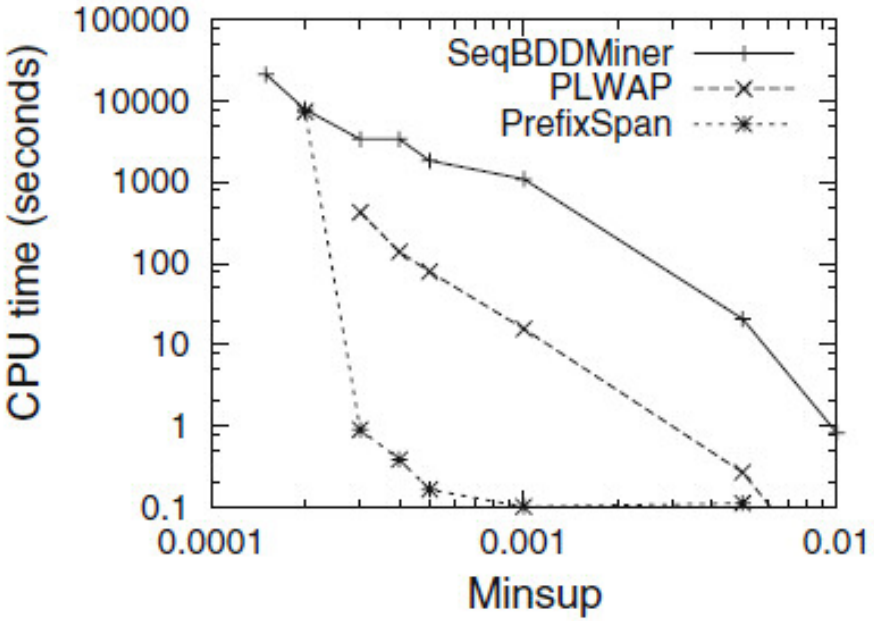


(b) DENV1

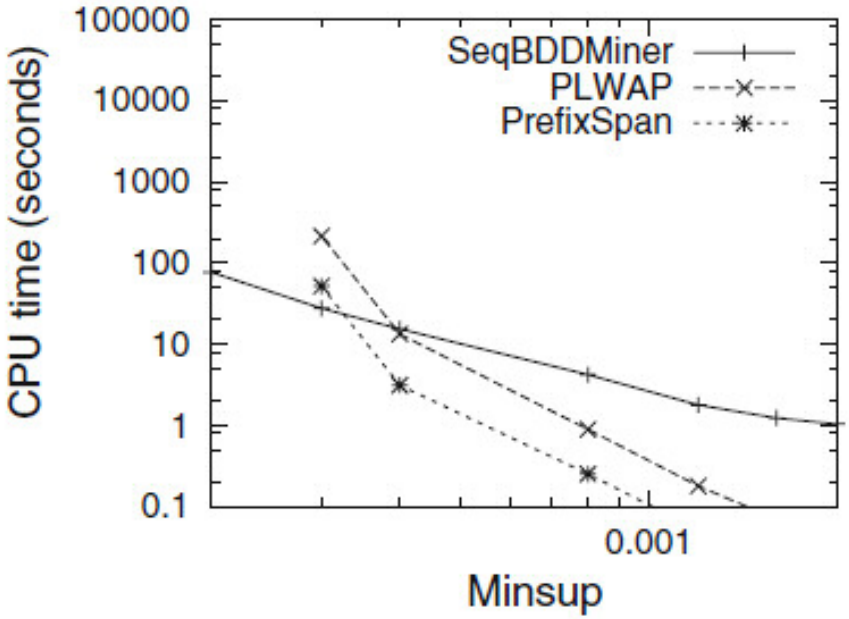
Performance study

■Runtime performance of the mining algorithm

- gazell(a) and davinci(b)データセットではPLWAPとPrefixSpanの計算速度の優位性が確認される．ただし閾値が下がった際に最も計算速度の増加が緩やかなのはSeqBDDMiner.
- Gazellデータセットは閾値が非常に小さくなった際にSeqBDDでは計算ができなくなる.



(a) gazell

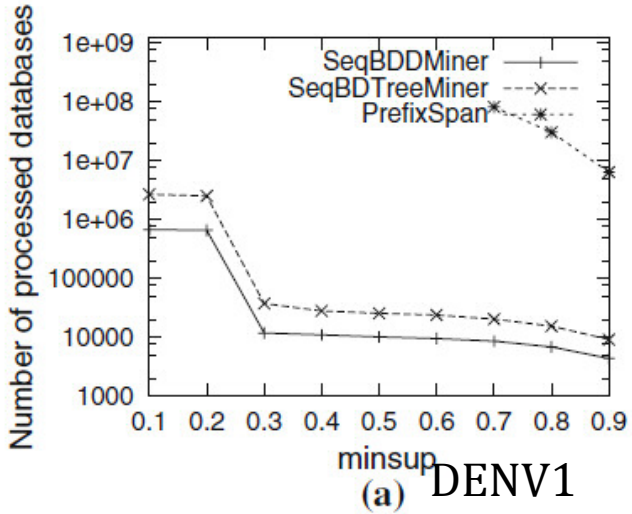


(b) davinci

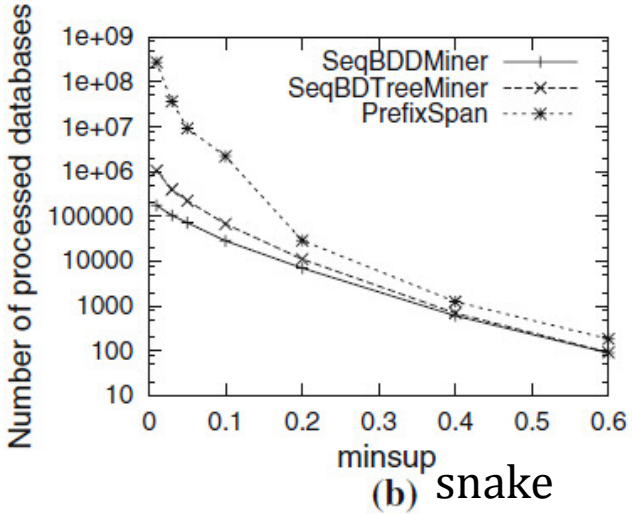
Performance study

■Effectiveness of SeqBDDMiner due to pattern caching and node sharing

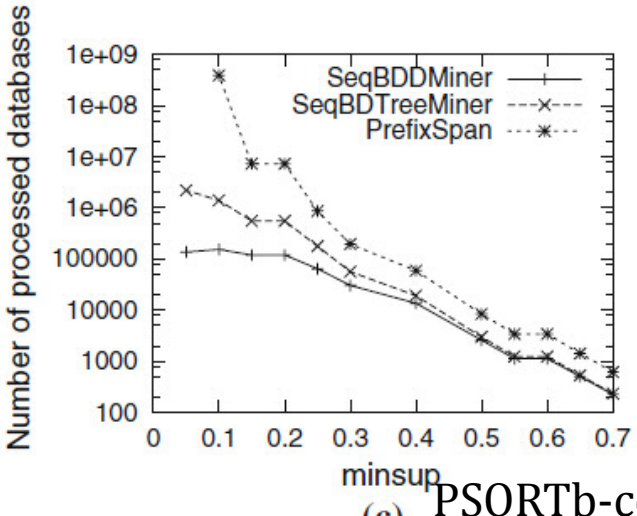
- SeqBDDMinerはどのデータについても最も小さいキャッシュ量で計算が可能．特に閾値が低い場合にその効果が大きい.



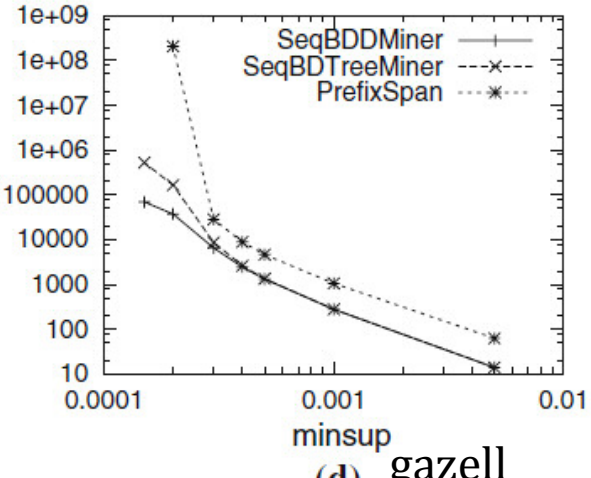
(a) DENV1



(b) snake



(c) PSORTb-com

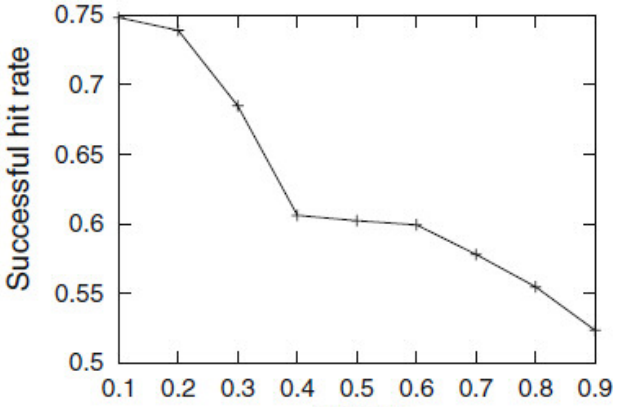


(d) gazell

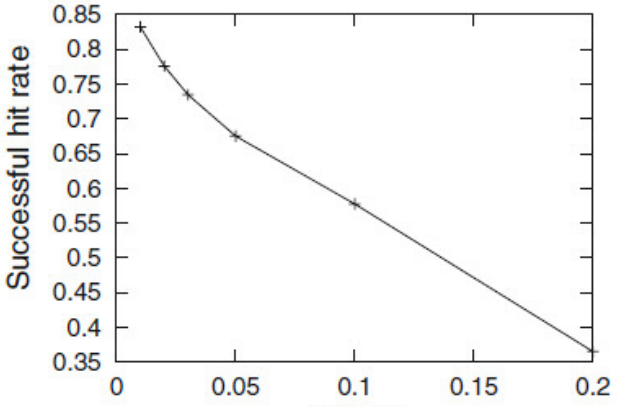
Performance study

■Effectiveness of SeqBDDMiner due to pattern caching and node sharing

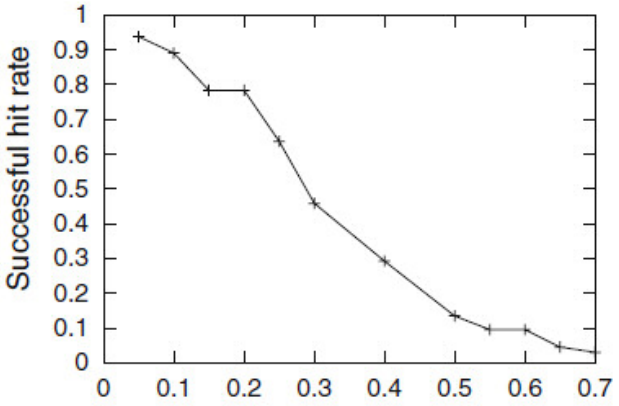
- 基本的にいずれも閾値を上げると的中率が下がる.
- Gazelle(d)は閾値を少し下げただけでも的中率が大きく低下する.



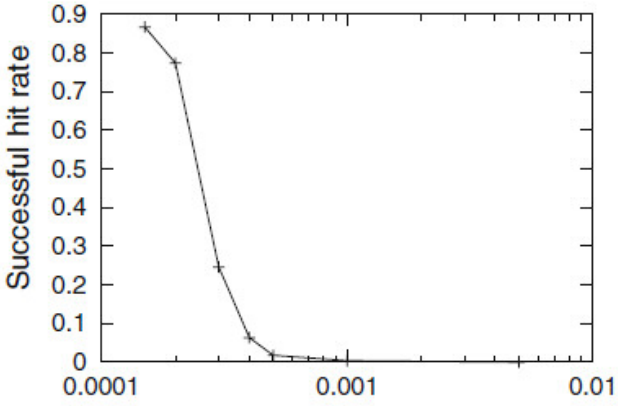
(a) DENV1



(b) snake



(c) PSORTb-com



(d) gazell

Discussion

■Highly similar sequences

- 系列間の類似性が高いデータの場合，SeqBDDでノードの共有が行われるメリットが大きい．特に系列が長い場合その効果が大きい．
- 例えばDNAやたんぱく質の配列のデータセットにおいて，パターンキャッシュの的中率が高く，ランタイムのパフォーマンスも高い．
- ただし，類似性が高くても系列が短い場合は二分木のノードの共有化のメリットが少ないため，PLWAPやPresfixSpanのほうが優位性が高いこともある．

■Dissimilar sequences

- 類似性が低いデータの場合，SeqBDDでのノードの共有が少ないため，導入のメリットが少ない．
- しかし，頻出閾値が低い場合の計算を行う場合にはメリットがある．

Discussion

BDDを用いて系列をコンパクトに表現することはできるか？

- 系列を扱うことができるSeqBDDを用いることにより，変数順序の制約があるZDDの半分ほどのデータ量で計算が可能になる．
- ノードがどれだけ共有できるかは初めに構築したinitial SeqBDDのデータ量に依存する．

頻出パターンマイニングにおけるBDD利用のメリットは？

- 演算処理ができることとキャッシュを用意できる点がポイント．
- キャッシュを用意することで再帰的な呼び出しをせずに即座に結果を返すことができ，演算処理ができるのでサブツリーに分割して演算することができる．

BDDを用いた頻出パターンマイニングはパターン成長アプローチの最先端となり得るか？

- インプットする系列が長いもしくは類似性が高い場合，SeqBDDMinerはPLWAPやPrefixSpanを上回る性能を示す．
- しかし系列が短いもしくは類似性が低い場合，ノードが共有できる点のメリットが小さいので，閾値が小さい場合のスケーラビリティの高さ以外のメリットに乏しい．

所感

- 時系列データを扱おうとすると系列の考え方は必須になるので通常のZDDでは対応しきれない．また，ルートの重複を許すZDDを構築しようとするとネットワークを冗長化させることになるのでデータの制約が大きい．
- ということで，少なくとも系列を扱うのであればZDDよりSeqBDD．
- SeqBDDは上記の問題をカバーできそうだが，クロスセクションで適用しようとすると，まずは運行経路の実データが無いとBDDの構築が難しい（構築できないことはないがランダム系列だと圧縮効率が低い）．
- 毎日の行動データをSeqBDDでストックしておき，その類似性を共通部分列や部分文字列を整理することで把握→運行ルートの検討や乗客のまとめ方を検討するような使い方はできそう．
- 単純に時空間ネットワークにするだけなら，各時点でZDDを構築して足し合わせる方が手っ取り早いかもしれない（ルート重複の問題は残るが・・・）．
- いずれにしてもネットワーク規模が大きくなる場合は単一のZDDではなくZDDの集合演算で処理しないと現実的な時間内での計算が不可能になると思われる．

その他参考にした文献

- ERATO湊離散構造処理系プロジェクト: 超高速グラフ列挙アルゴリズム, 森北出版株式会社, 2015.
- Loekito, E. and Bailey, J.: Are zero-suppressed binary decision diagrams good for mining frequent patterns in high dimensional datasets?, *AusDM '07 Proceedings of the sixth Australasian conference on Data mining and analytics*, Vol.70, pp. 139-150, 2007.
- Kurai, R., Minato, S., Zeugmann, T.: N-gram analysis based on zero-suppressed BDDs, *New Frontier in Artificial Intelligence*, Vol. 4834, pp. 289-300, 2007.
- 伝住周平, 有村博紀, 湊真一: 系列二分決定グラフを用いた部分文字列索引の構築, 第2回データ工学と情報マネジメントに関するフォーラム, E3-4, 2010.

SeqBDD

■SeqBDDの構造

- SeqBDDの節点$N = node(x, N_0, N_1)$は文字xを持つ内部節点を示し，$N_0(N_1)$は1-枝（0-枝）が指す節点を示す．節点NはN_0よりも変数順序で先に現れる．
- 節点Nの子孫節点の合計数をN自身も含めて$|N|$で示す．
- xをある文字，PとQを二つの系列集合とする．
- xをP中の全系列の先頭に付加する演算として$x \times P$とし，PもしくはQに出現する系列の集合を返す演算として$P \cup Q$を定義する．

SeqBDD

■論理関数

a	b	c	F	
0	0	0	0	
0	0	1	1	$\longrightarrow c$
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	$\longrightarrow ac$
1	1	0	1	$\longrightarrow ab$
1	1	1	0	

論理関数とみた場合：

$F = a\,b\,\bar{c} + \bar{b}\,c$

組合せ集合とみた場合：

$S = \{ab, ac, c\}$

図 10.5　論理関数と組合せ集合の対応