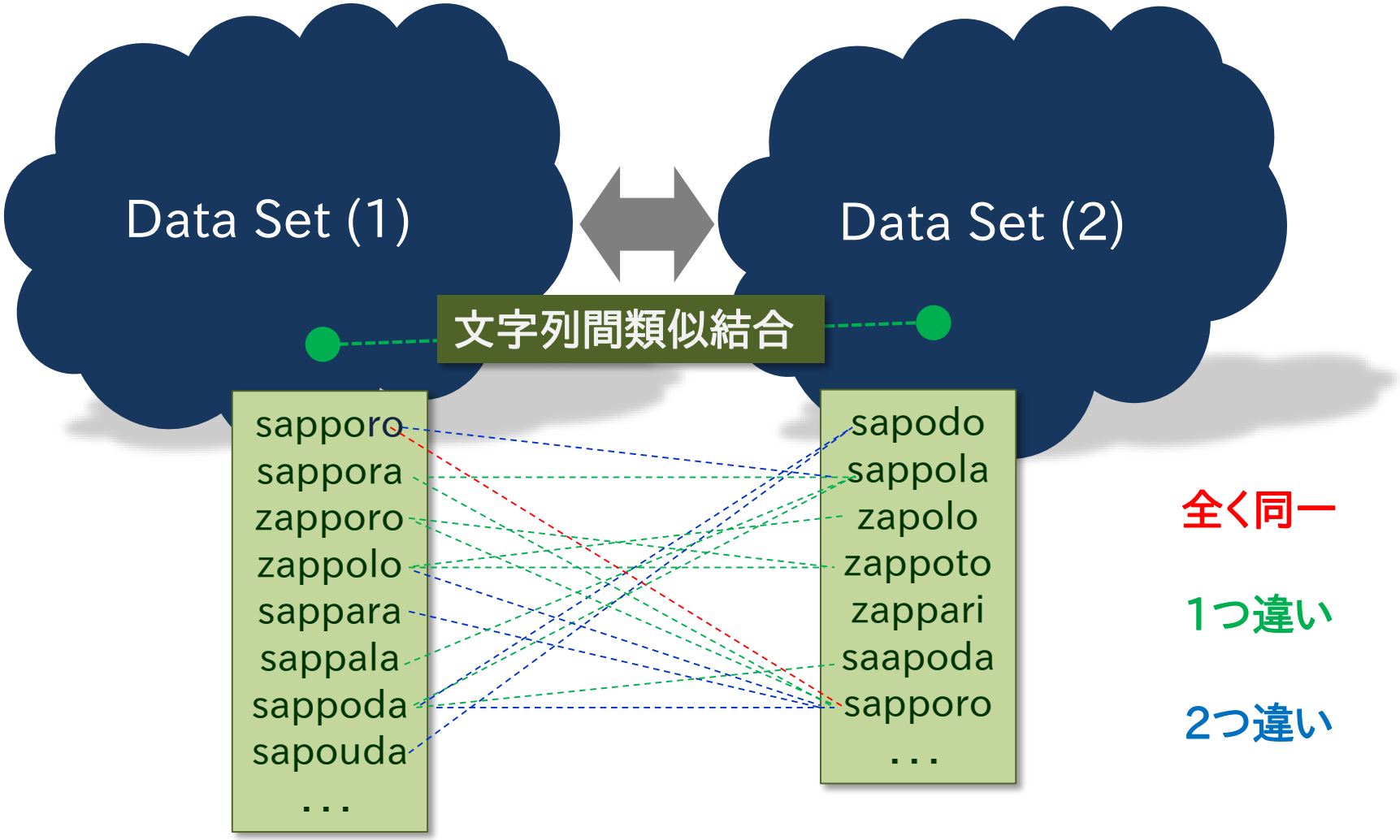
SeqBDDを使用した文字列間類似結合

高嶋宏之, 白井康之

（独）科学技術振興機構
ERATO 湊離散構造処理系プロジェクト

takashima@erato.ist.hokudai.ac.jp

背景



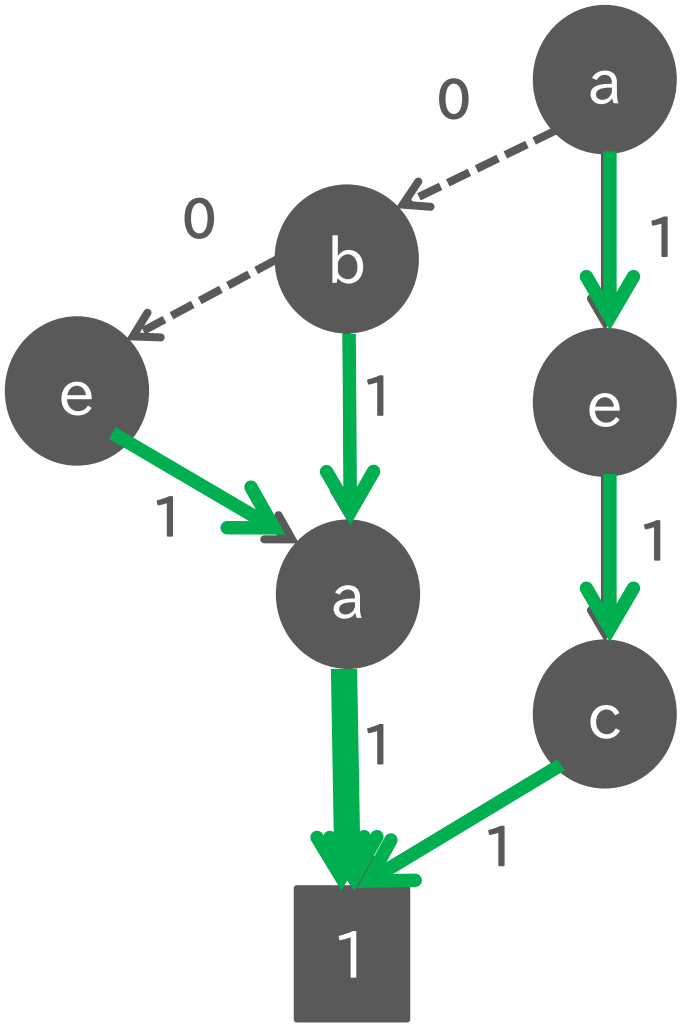
→ 一定の条件下（編集距離等）でマッチする類似ペアを**すべて**列挙する.

背景

- フィルタリング手法
  - 分割・枝刈り
  - q-gram インデクシング
  - prefix / suffix によるフィルタリング
- データの構造化に基づく手法
  - Trie構造を用いた手法（Trie-Join）　ツリー構造
  - SeqBDDを用いた手法（SeqBDD-Join）　DAG構造

	フィルタリング手法	データの構造化に基づく手法
レコード長	長い（ほうが得意）	短い（ほうが得意）
アイテム種類	多い	少ない
制約（編集距離）	大きい	小さい

ZDD、SeqBDDについて



SeqBDD(SequenceBDD):
ZDDをSequencuialに拡張したもの。

0枝はアルファベット順

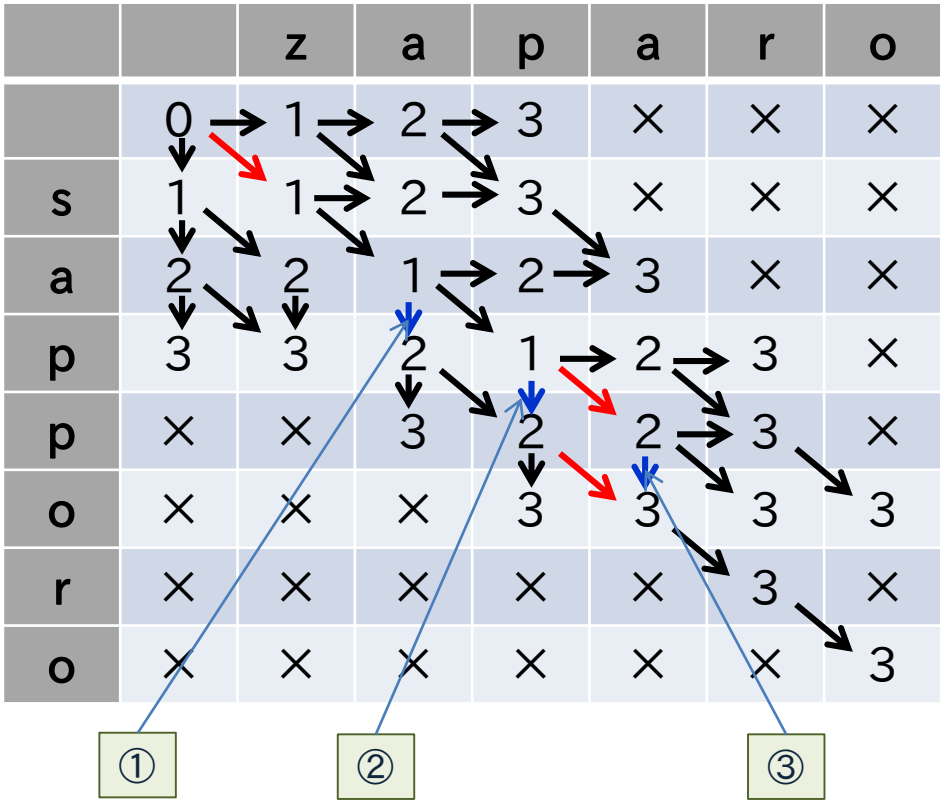
$\{ aec, \quad ba, \quad ea \}$

探索手法(動的計画法)

動的計画法によるストリングマッチング

sapporo

zaparo　　(制約 : *Edit Distance=3*)



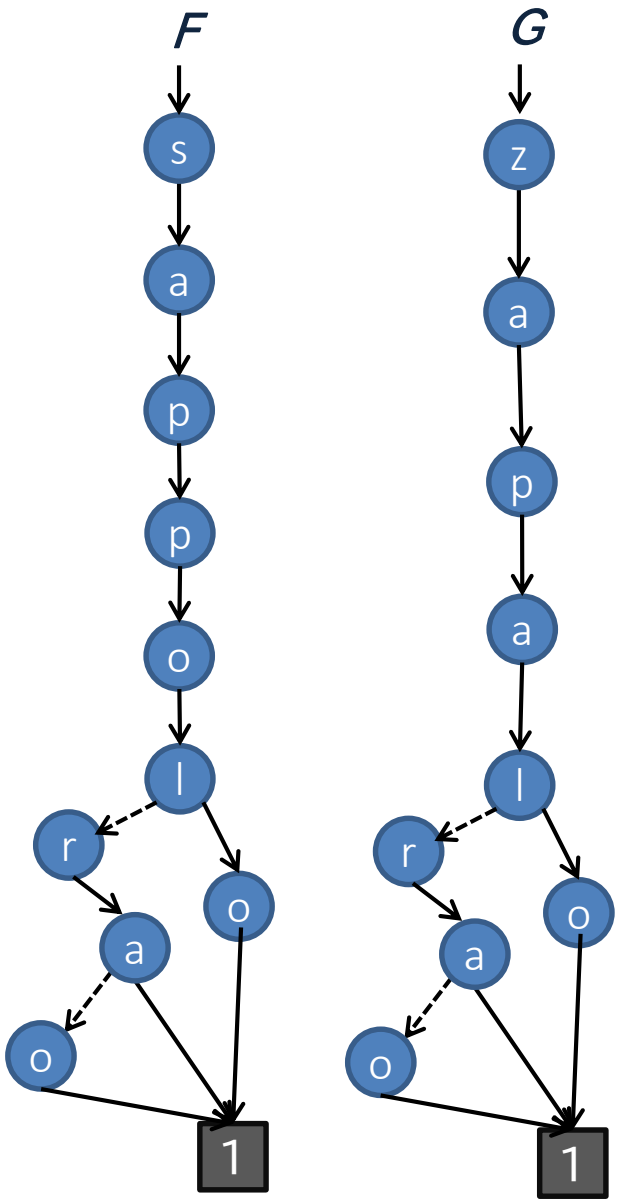
① sa p p o ro
　 za　 p a ro

② sa p p o ro
　 za p　 a ro

③ sa p p o ro
　 za p a　 ro

(赤字は replace, 青字は insert/delete)

探索手法(動的計画法)



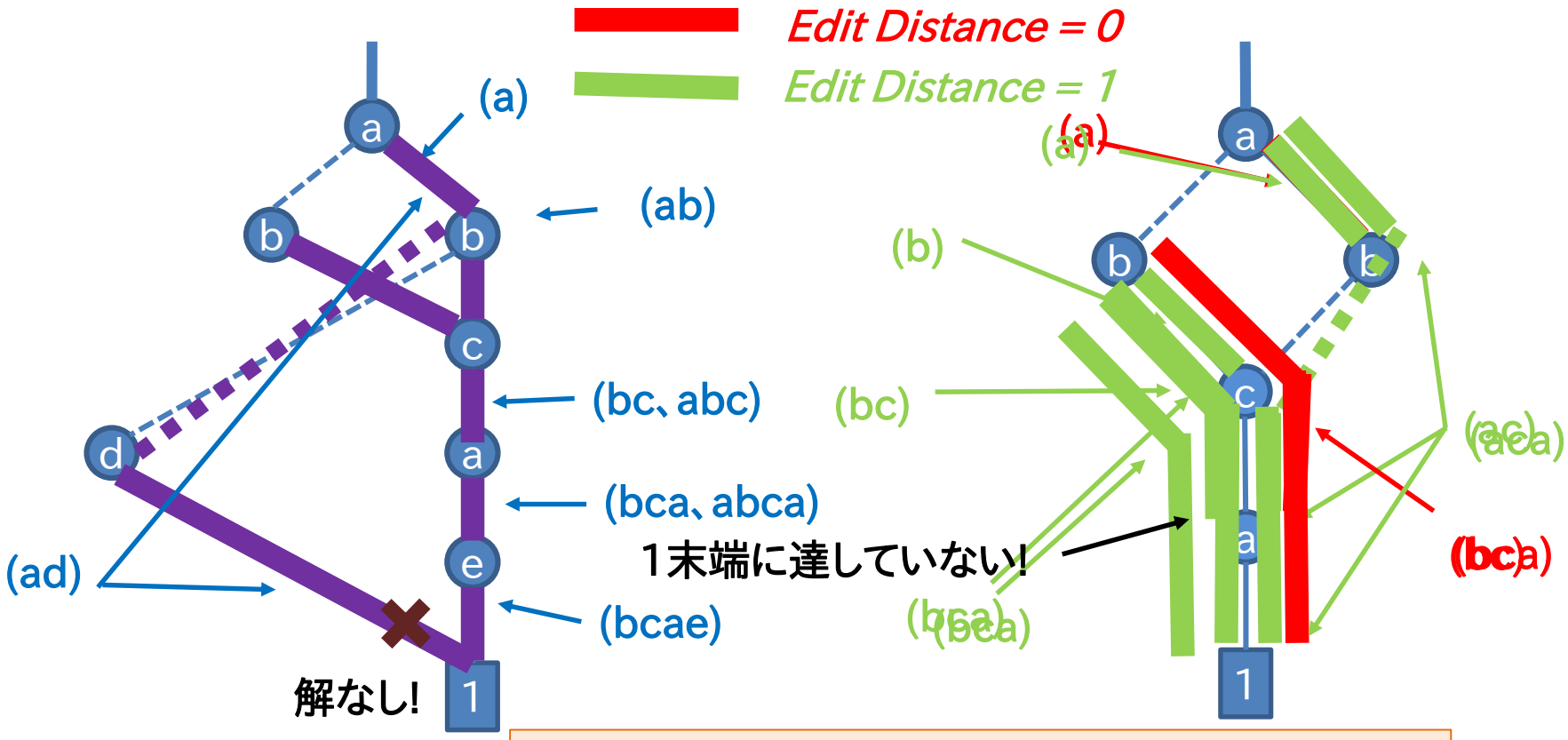
F: sappolo, sappora, sapporo
G: zapalo, zapara, zaparo

F \ G				z	a	p	a	r	o
								r	a
								l	o
			0	1	2	3	×		
		s	1	1	2	3	×		
		a	2	2	1	2	3		
		p	3	3	2	1	2		
		p	×	×	3	2	2		
		o	×	×	×	3	3		
r	r	l							
o	a	o							

探索手法(動的計画法)

データベースF　　制約：*Edit Distance = 1*　　*データベースG*

{ abcae, ad, bcae }　　{ aca, bca }



データベース*F*　側の“bcae”のパスに対し、*G*側にEdit Distance = 1の解が一つ存在!!

実装上の工夫1(最小距離増分による枝刈り)

制約 Edit Distance = 3, {sapzapo、zapo}

両文字列間で3(7-4)文字異なることは明らか。

→ **最小距離増分**と定義。

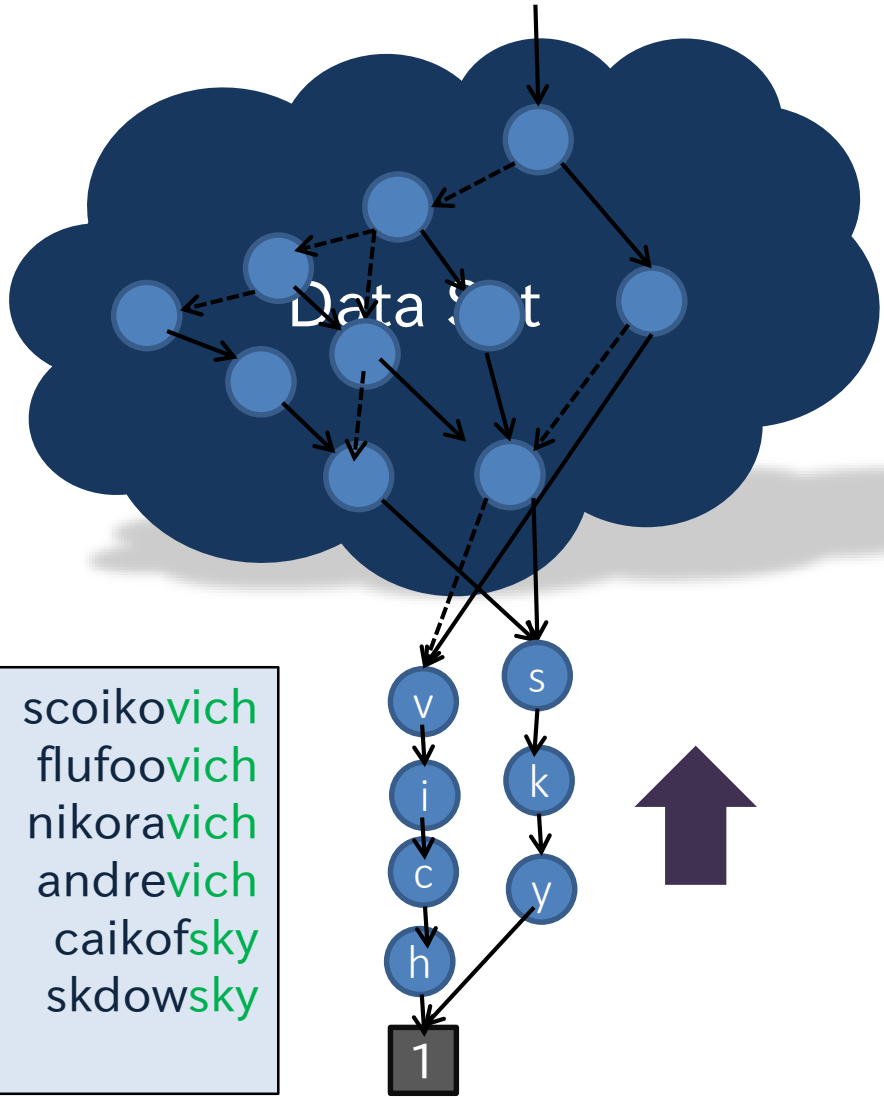
		z	a	p	o
	5	4	3	2	1
8	0	1	2	3	×
7 s	1	1	2	3	×
6 a	2	2	1	3	×
5 p	3	3	2	1	2
4 z	×	3	3	2	2
3 a	×	×	3	3	3
2 p	×	×	×	3	×
1 o	×	×	×	×	3

編集距離4(1+3)に相当し、枝刈り可能!!

		z	a	p	o
	0	1	×	×	×
s	1	1	×	×	×
a	2	2	×	×	×
p	3	3	×	×	×
z	×	3	3	×	×
a	×	×	3	3	×
p	×	×	×	3	×
o	×	×	×	×	3

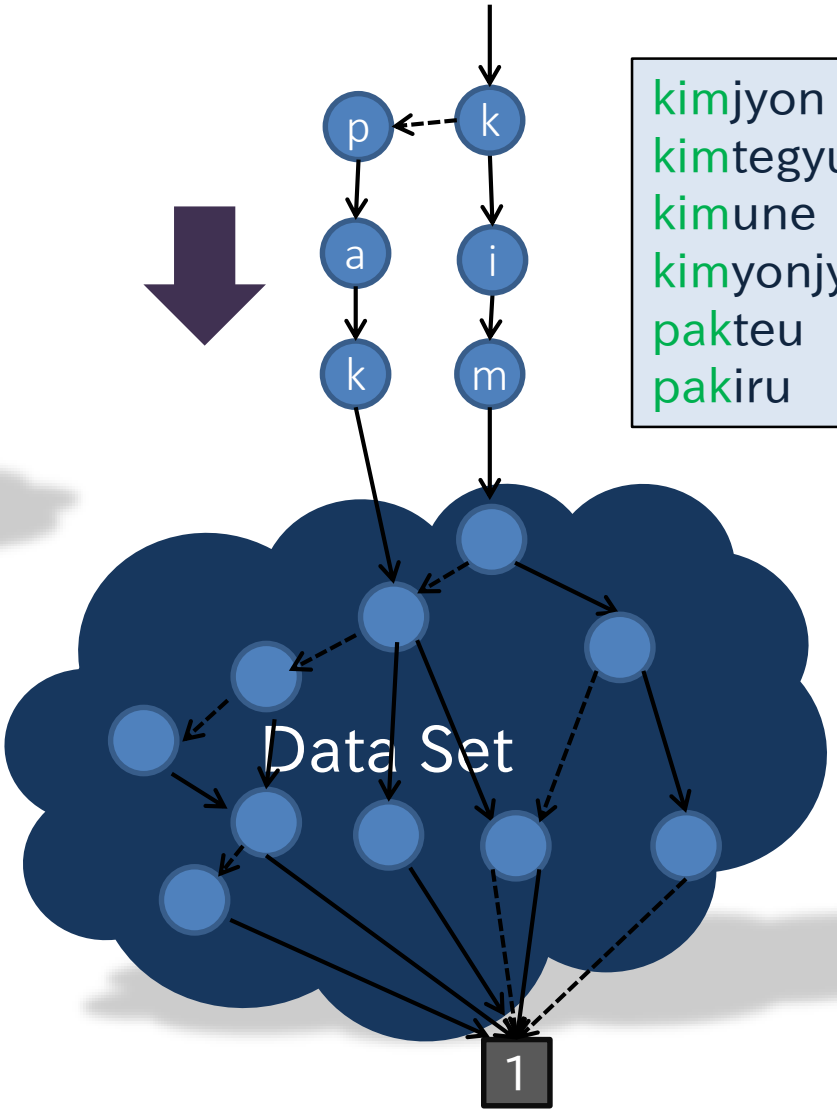
実装上の工夫2(Hybrid 型による探索)

例:東ヨーロッパの人名



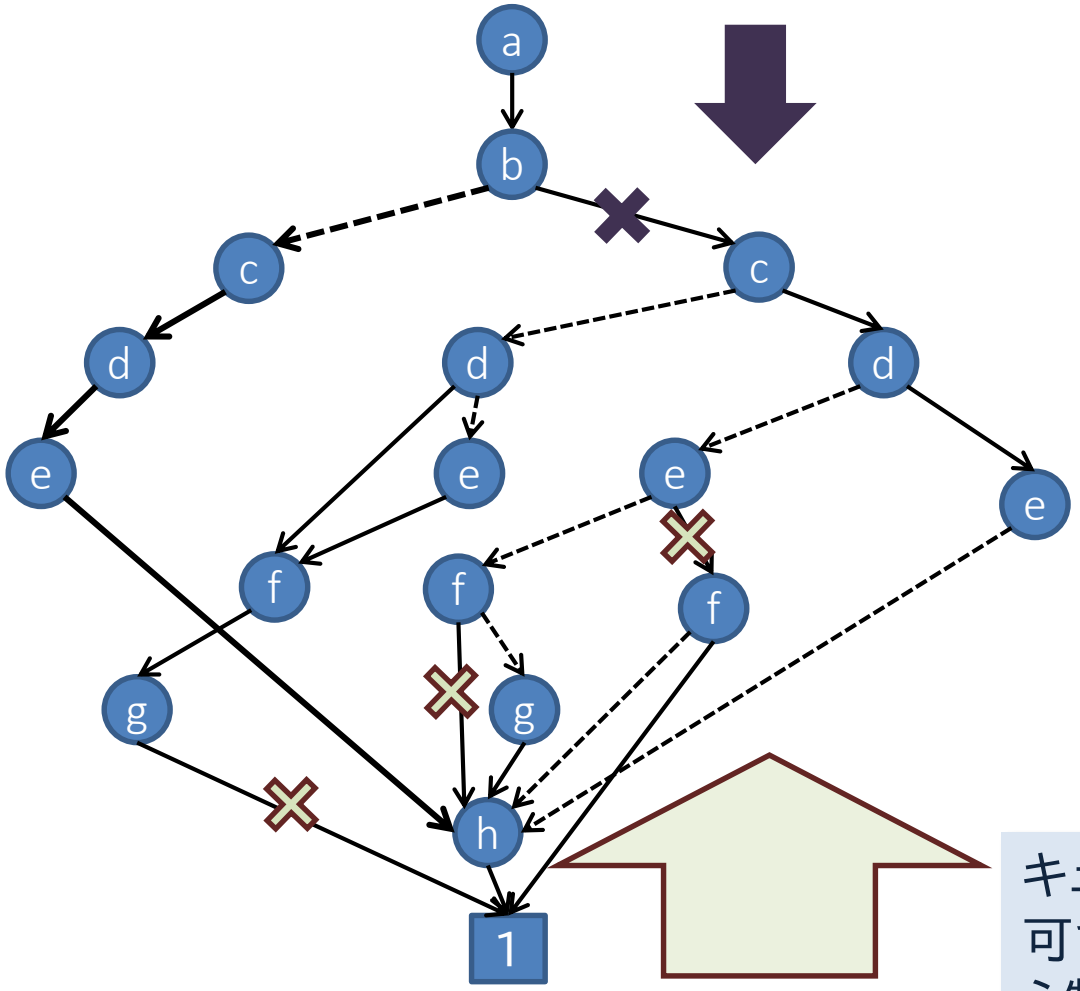
scoikovich
flufoovich
nikoravich
andrevich
caikofsky
skdowsky

例:韓国の人名



kimjyon
kimtegyun
kimune
kimyonjyu
pakteu
pakiru

実装上の工夫2(Hybrid 型による探索)



⇔

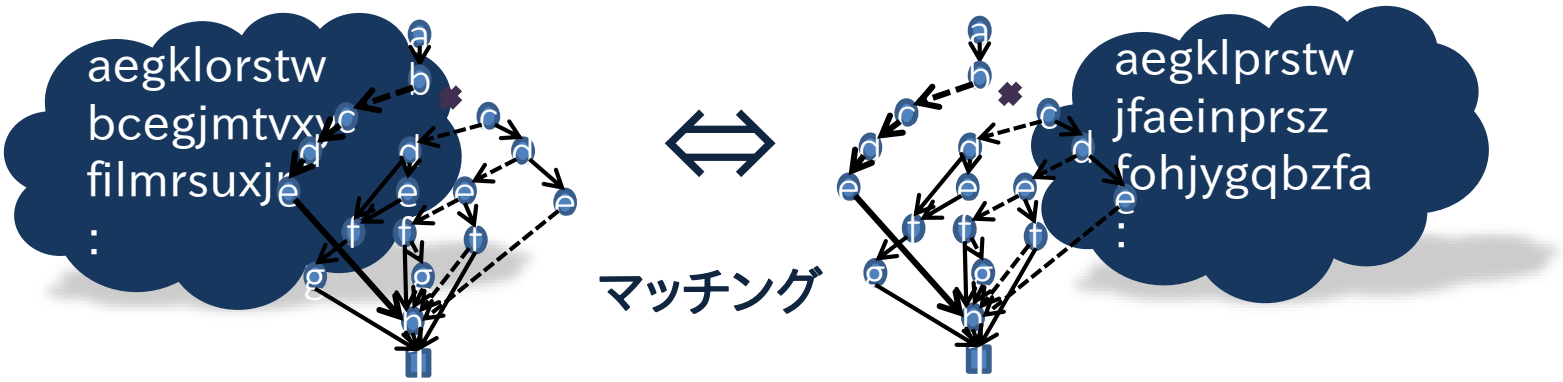
マッチング

Data Set

キューを用意し、早めに枝刈りできる可能性が高いノードから探索を行うよう制御する。

実験結果（工夫1の実験結果：最小距離増分による枝刈り）

データ：任意の10文字のアルファベットの文字列を用意し，100万×100万件のデータのマッチングを行った．

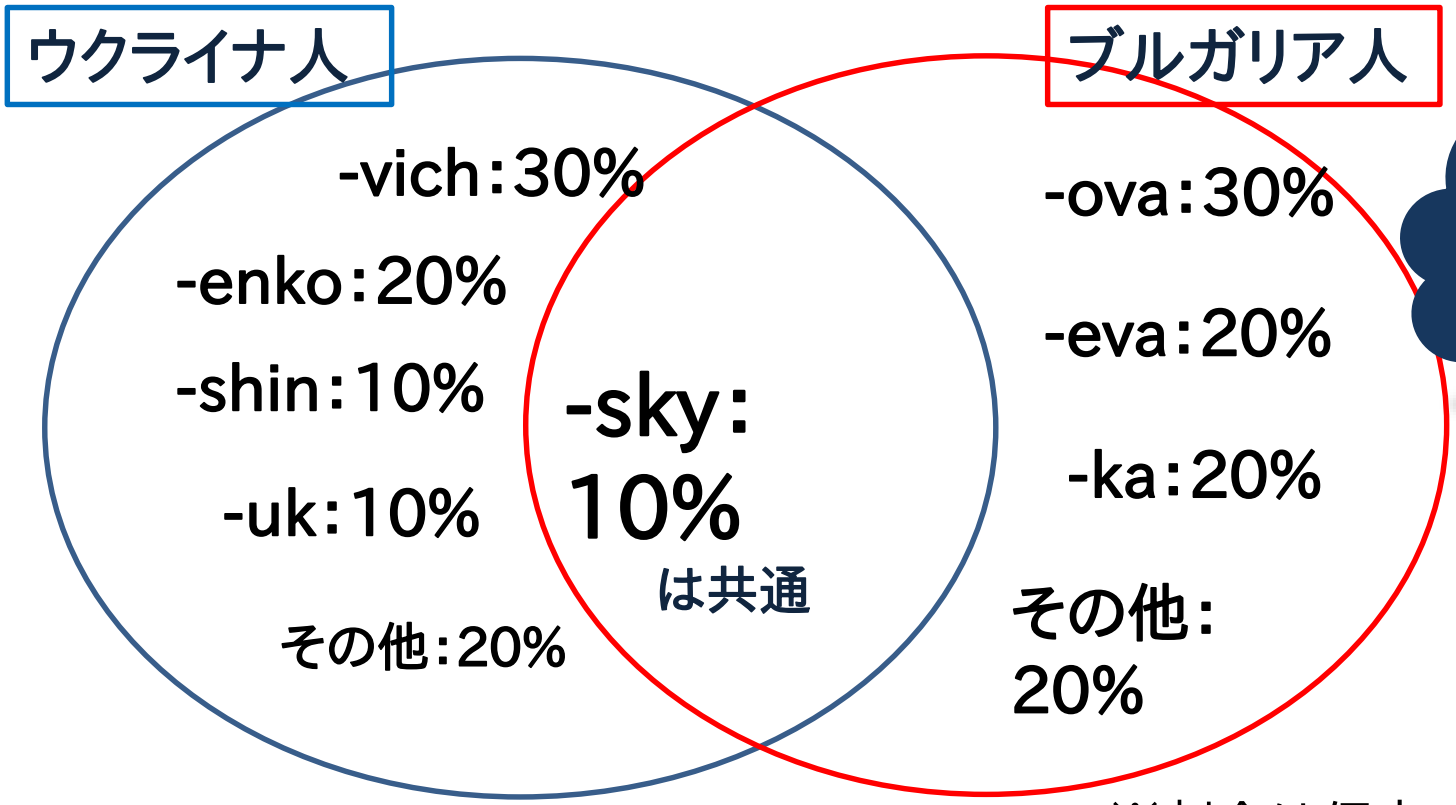


パターン	探索時間(秒)
最小距離増分考慮なし	149.5
最小距離増分考慮あり	119.3

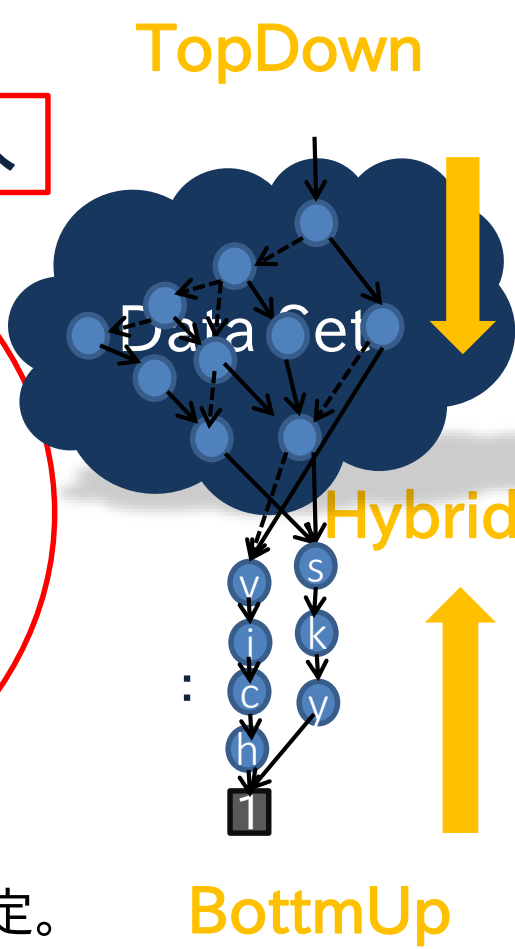
→ 枝刈りの効果があり、**約20%**探索時間が改善された。

実験結果（工夫2の実験：Hybrid 型による探索）

東ヨーロッパの国間の氏名のマッチング



※割合は仮定。



人名（東ヨーロッパ）より
http://www.asahi-net.or.jp/~ax2s-kmtn/ref/pname/e_europe.html

実験結果（工夫2の結果：Hybrid 型による探索）

10万×10万の10文字のランダムな文字列をベースに一部加工

①：東ヨーロッパの人名(-sky, -vichなど**接尾辞**のみ変更)

②：韓国の人名(kim-,pak-など**接頭辞**のみ変更)

③：ランダムデータ

パターン	①	②	③
TopDown 型	80,470,576	50,364,085	80,522,604
BottomUp 型	24,465,874	158,112,786	80,413,800
Hybrid 型	54,649,687	60,012,881	113,361,424

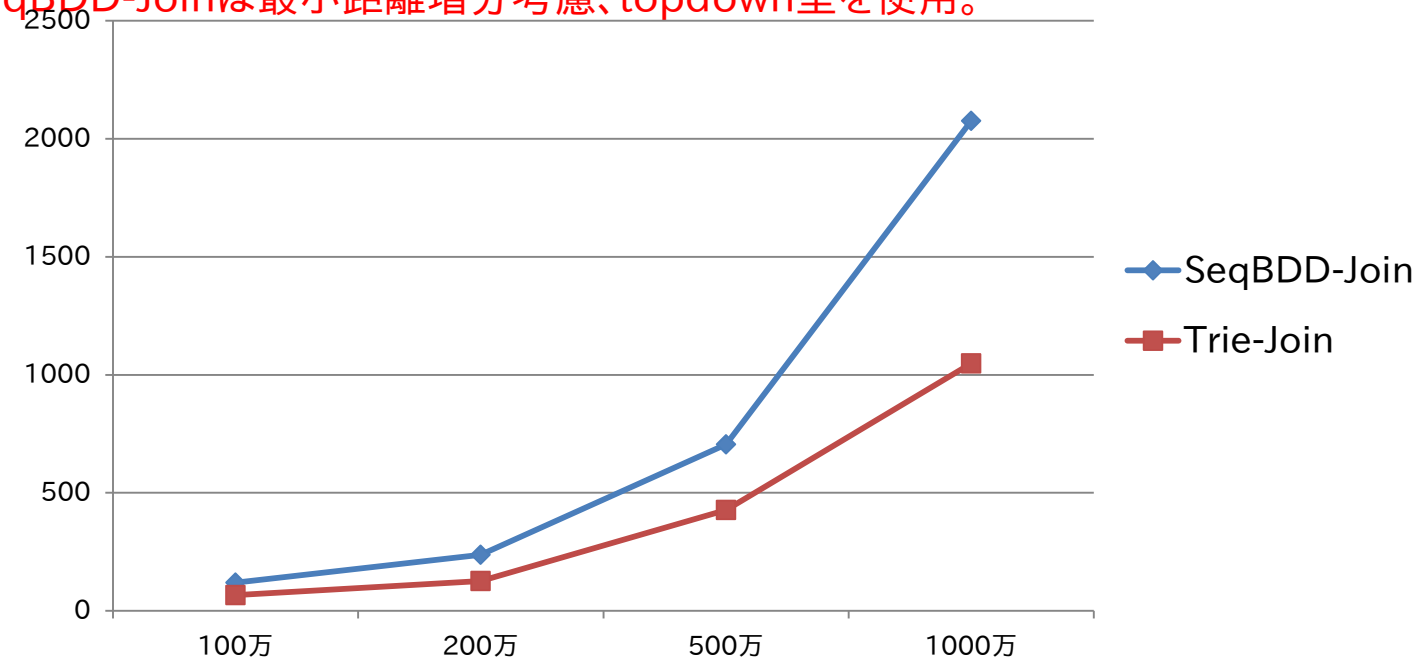
※動的計画法のcall数で比較。

→①、②のような比較的偏りが多いデータの場合、ワーストケースの場合の実行時間の最小化という観点では、Hybrid型手法で探索を行った方が効果的。

実験結果(Trie-Join との比較)

データ: ランダムな10文字のアルファベットの文字列を100万×100万件、500万×500万件、1000万×1000万件を用意し、データベース間でマッチングを行った。

※SeqBDD-Joinは最小距離増分考慮、topdown型を使用。



結果)
現時点では、Trie-Joinに比べて1.5～2倍程度性能が悪い。

考察（性能について）

性能に影響する要因として、以下のようなことが考えられる。

1 プログラミングの問題

→ C++のクラスのメンバの定義、インターフェース
などにまだ不備がある？

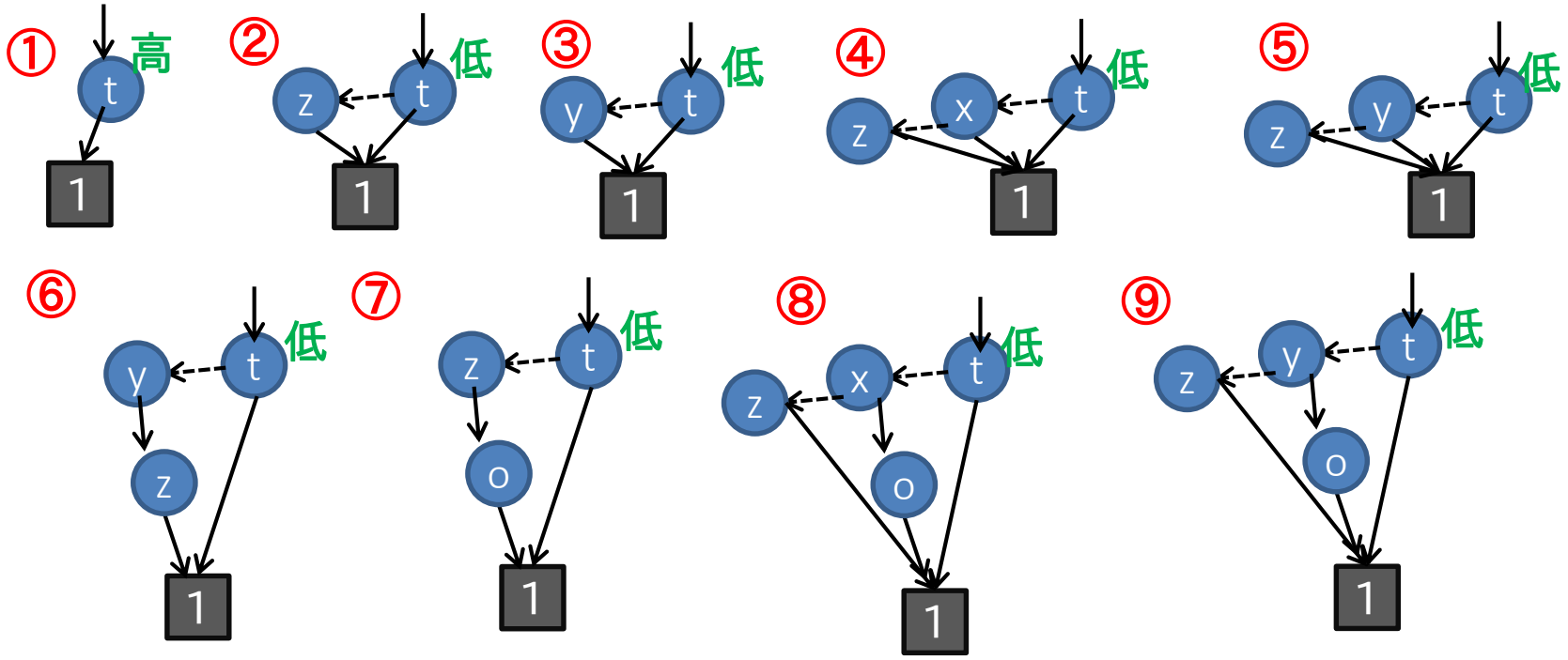
2 SeqBDDの構造が影響

→ SeqBDDはDAG構造のため、Trieと比べてノードが共有されていることが利点。しかし予想以上に共有されていないように見受けられる。

考察（SeqBDDの構造）

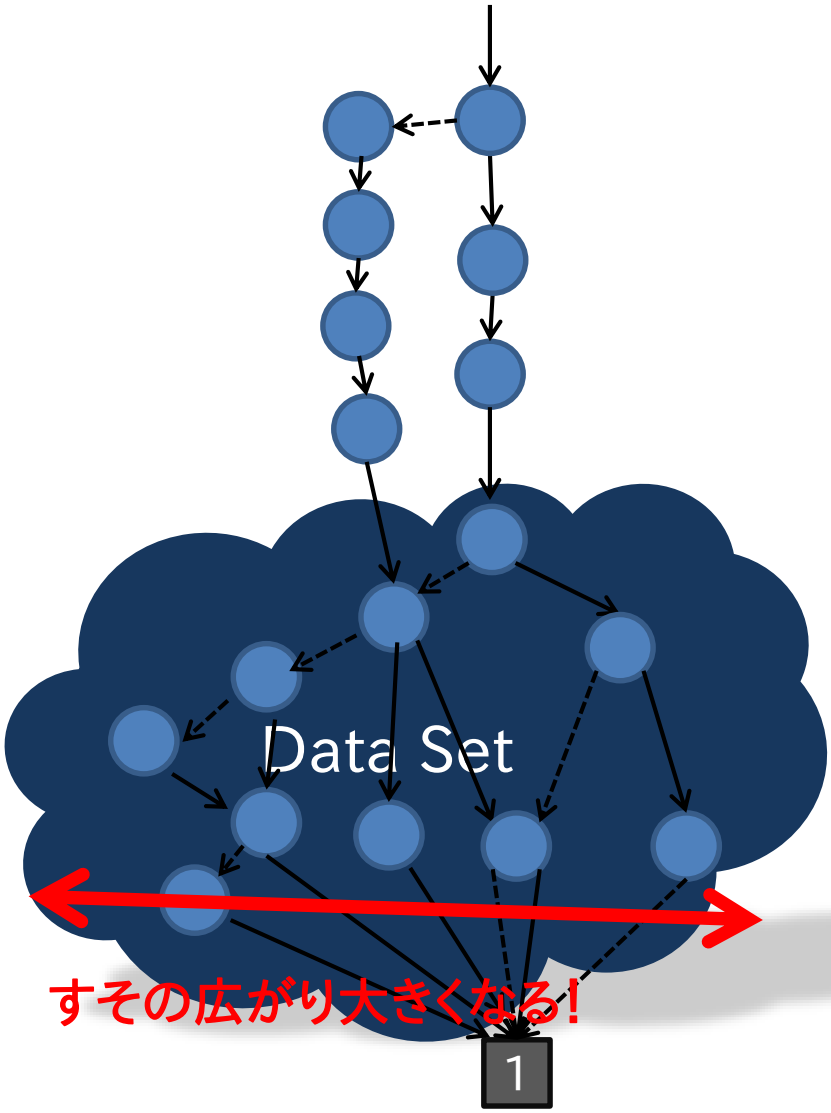
接尾辞が-tで終わるSeqBDDの構造例

※tの右上は
上位の共有率。



SeqBDDは0側を考慮した、様々な形のものが作成される。例えば、接尾辞がtで終わる文字でも上記のような例がある。①のようなtの0側が存在しない場合は上位で比較的多く共有されているが、②～⑨のものは①よりは特殊な形のため上位で共有される割合は低くなる。
→②以降のようなものが作られれば作られるほど、トータル的に共有率の低いノードが増える。

考察（SeqBDDの構造）



前ページは“t”の文字の例だが、他の文字でも同様で、1末端に近いノードはあまり共有されていないノードが増える可能性が高い。

⇩

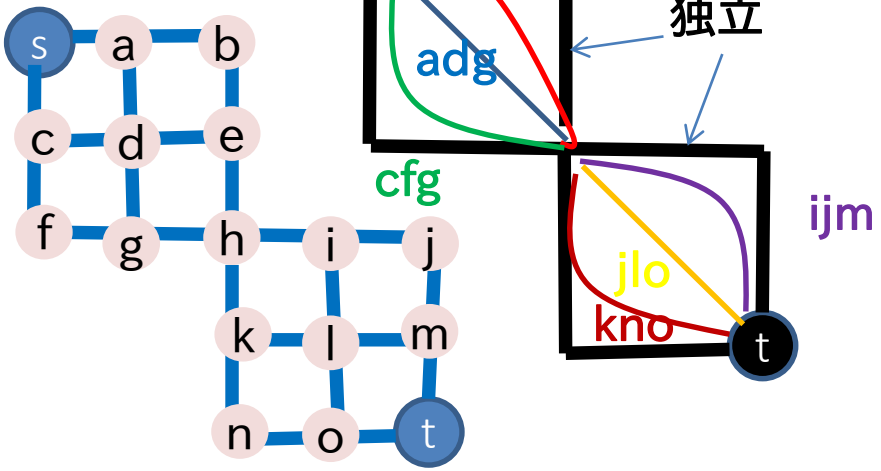
あまり共有率は高くないなら、1末端に近づくにつれすその広がりが大きくなり、
DAG→Tree構造に近づいている。

⇩

SeqBDDで本当に効果的に共有されているのか？

SeqBDDは任意の文字列間の類似結合には向かないか？

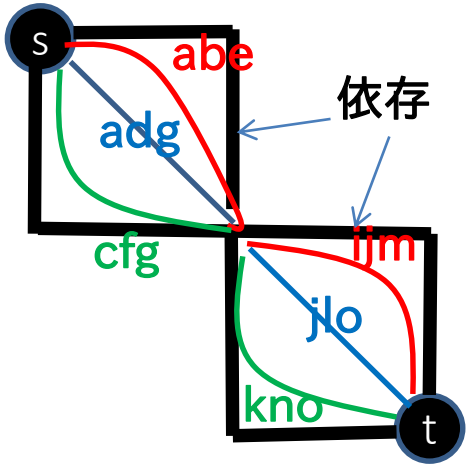
考察（SeqBDDの共有率）



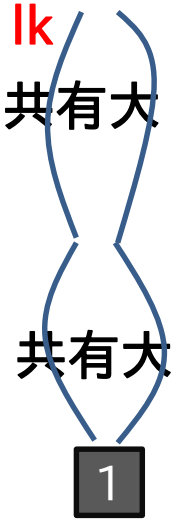
{a,b,e,i,j,m}

{a,d,g,i,l,o}

{c,f,g,k,n,o}

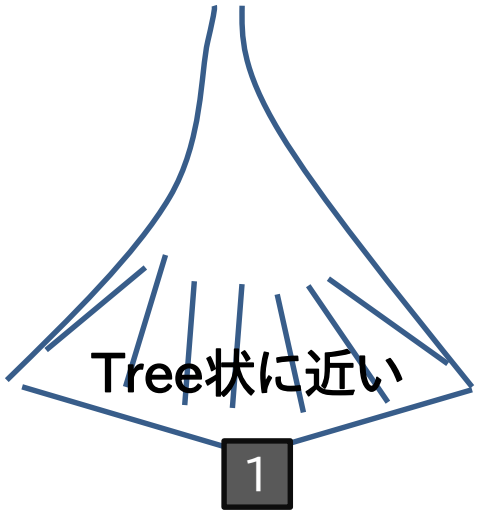


self_avoiding_walk



左の状態から独立性を崩すと、
右のTree状に近づいていく。
⇒次ページ。

任意の文字列間類似結合



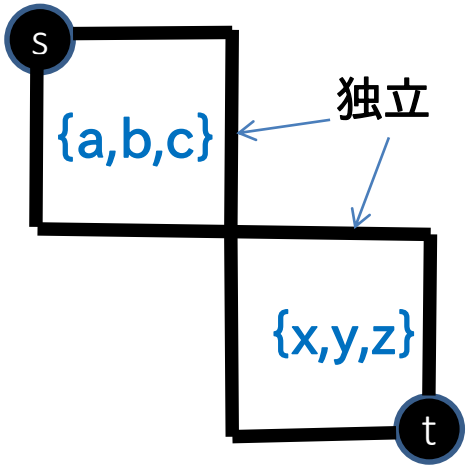
考察（SeqBDDの共有率）

追加実験：
考えうる全ての組み合わせをSeqBDDで表現
15通り×15通り ＝225通り

a,b,c,ab,ba,ac,ca,bc,cb,abc,acb,bac,bca,cab,cba

×

x,y,z,xy,yx,xz,zx,yz,zy,xyz,xzy,yxz,yzx,zxy,zyx

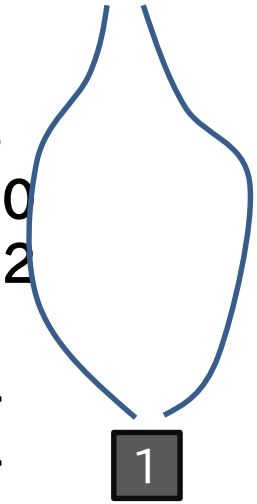


独立性を徐々に崩してゆく
⇒ 少し変えただけで一気にTreeに近づく。

ノード数
1階層：3
2階層：6
3階層：3
4階層：3
5階層：6
6階層：3
合計：24

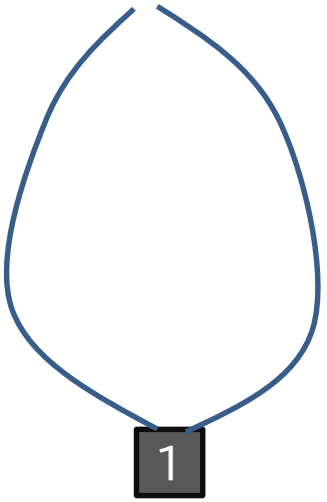
225個から
5個削除
2.2%(全体)
⇒

ノード数
1階層：3
2階層：6
3階層：10
4階層：12
5階層：9
6階層：4
合計：44



220個から
5個削除
4.4%(全体)
⇒

ノード数
1階層：3
2階層：10
3階層：12
4階層：14
5階層：12
6階層：4
合計：55



まとめ、FutureWork

【まとめ】

- SeqBDDをベースに、動的計画法を使用して文字列マッチングを行うプログラムを実装した。
- 明らかに文字数が異なる箇所に着目するなどして、最小距離増分と呼ぶ手法を行い20%程度枝刈りできた。
- 上位からの探索に加えて、下位からも探索を行うHybridの手法を実装し有効に働く可能性のある例を示した。

【FutureWork】

- 他の類似手法(Trie-Join)との比較。
- ランダムデータに対するHybrid手法(探索制御)の適用。
- SeqBDD(ZDD)を使用した集合演算への適用。