系列二分決定グラフを用いたウォークの列挙

青木洋士[*1]

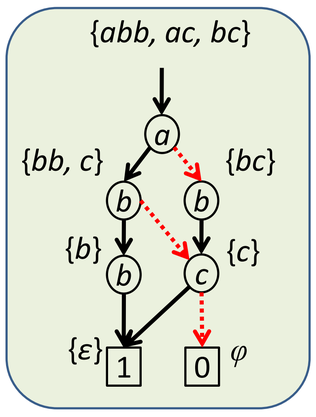
[*1] Graduate School of Information Science and Technology, Hokkaido University

系列二分決定グラフ (SeqBDD) [1]

・系列の集合を表現
・共通な接頭辞・接尾辞を共有

各パスが系列を表現
実線の1-枝を通ると、その文字が系列に含まれる
破線の0-枝を通ると、その文字が系列に含まれない

部分グラフ系列の集合を表現する



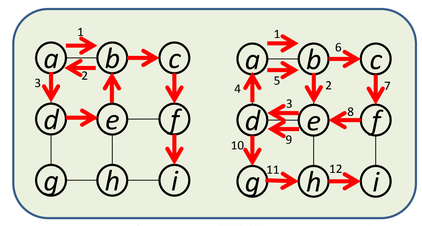
SeqBDD

ウォーク列挙問題

グラフ上から特定の条件を満たすウォークを列挙したい

・**自己回避ウォーク** (Self Avoiding Walk, SAW)
同じ節点を高々1度しか通らないウォーク
正方グリッドグラフの自己回避ウォークは、おねえさん問題[2]

・**k-自己不回避ウォーク**
同じ節点を高々k度しか通らないウォーク
SAW = 1-自己不回避ウォーク



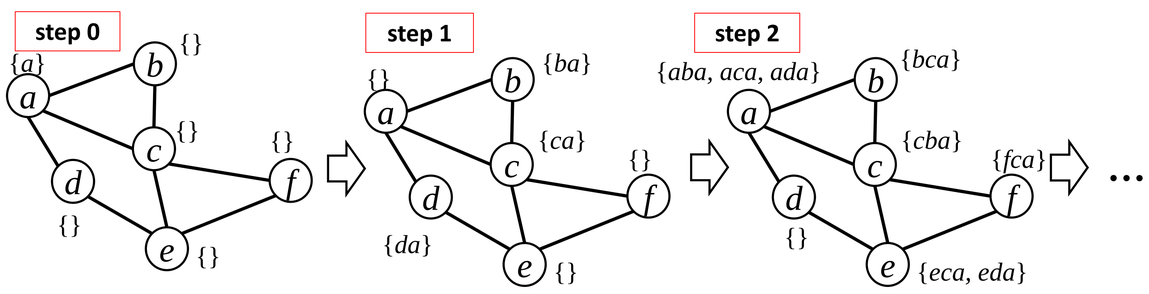
2-自己不回避ウォーク

・**t-ステップウォーク:**
始点からt-ステップでちょうどで到達するウォーク

準備

Walks(t): 全てのt-ステップウォークの集合を表すSeqBDD

・各節点にウォーク集合を表すSeqBDDを対応付け
・t-ステップのシミュレート
・終点に対応付けられたSeqBDDが、終点までのt-ステップウォーク集合に対応



P.avoidAt(v, n): ウォーク集合の節点vでのk-自己不回避化

ウォーク集合を表すSeqBDD Pについて
・節点vをn回以下しか通らないウォークのみを含むようにする

```
P.avoidAt(v, n) := {
 if (F == TOP || F == BOT) return P;
 if (F.top == v) {
  if (n == 0)
   Q1 ← BOT;
  else
   Q1 ← F.advoidAt(F1, v, n-1);
 }
 else
  Q1 ← F.avoidAt(F1, v, n);
 Q0 ← F.avoidAt(F0, v, n);
 return Q1.Push(F.top) U Q0;
}
```

$\{abad, abcb, abcd\}$ .avoidAt(a, 1)
$= \{abcb, abcd\}$

k-自己不回避ウォークを表すSeqBDDの構築

naive手法 n-自己不回避化を節点数回適用

Result ← Walks(t) .avoidAt(v_1 , n) .avoidAt(v_2 , n) …

apply手法 1 n-自己不回避化したウォーク集合の積集合をとる

A_i := Walks(t) .avoidAt(v_i, n);
Result ← $A_1 \cap A_2 \cap A_3 \cap A_4 \cap \ldots \cap A_n$

apply手法 2 手法1の積集合をとる順番を工夫する

$(A_1 \cap A_2) \cap (A_3 \cap A_4) \cap \ldots (A_{n-1} \cap A_n)$
$(B_1 \cap B_2) \cap \ldots \cap B_{n/2}$
…
Result

実験結果

・格子グラフの左上の節点から右下の節点までのn-自己不回避ウォークを表すSeqBDDを構築
・Ubuntu 12.04 64bit, gcc/g++ 4.6.3にて、SAPPORO BDDを用いて実装

ウォーク数の比較

格子サイズ	n :同じ節点を通れる数			
	1	2	3	4
1x1	2	22	246	2,990
2x2	12	9,482	11,659,388	Time out
3x3	184	112,269,228	Time out	
4x4	8,512	Time out		
5x5	1,262,816			
6x6	575,780,564			
7x7	Time out			

n=1の場合、フロンティア法のような効率は得られない

4x4, n=2のときの厳密解は知られていない [3]

実行時間の比較

格子サイズ	同じ節点を通れる回数											
	1			2			3			4		
	naive	apply1	apply2	naive	apply1	apply2	naive	apply1	apply2	naive	apply1	apply2
1x1	0.569	0.527	0.268	0.558	0.451	0.269	0.416	0.416	0.268	0.56	0.416	0.362
2x2	0.562	0.515	0.269	0.593	0.599	0.277	0.705	0.705	0.374	Time out	Time out	Time out
3x3	0.563	0.546	0.278	34.024	6.501	1.939	Time out	Time out	Time out			
4x4	0.843	0.665	0.358	Time out	Time out	Time out						
5x5	12.371	2.832	1.289									
6x6	Time out	72.697	73.132									
7x7		Time out	Time out									

上界と下界の計算

提案手法の途中打ち切り 途中で打ち切ると解の上界を取得できる
→ t-step ウォークから条件に合わないウォークを除いて行く手法であるため

Inclusion-Exclusion Methodに基づく数え上げ

上界と下界を狭めながら計算する

$|\cap A_i| = |\text{Walks}(t)| - \Sigma |A_i| + \Sigma |A_i \cap A_j| - \Sigma |A_i \cap A_j \cap A_k| + \ldots$

今後の課題

・途中計算打ち切りによる上界下界の計算
・アルゴリズムの高速化
・プログラムの並列化の検討
・他のパス列挙問題へのSeqBDDの利用

[1] E. Loekito, J. Bailey, and J. Pei, "A binary decision diagram based approach for mining frequent subsequences, " Knowledge and Information Systems: An International Journal, vol.24, no.2, pp.235 - 268, Aug. 2010.
[2] H. Iwashita, Y. Nakazawa, J. Kawahara, T. Uno, S. Minato, "Efficient Computation of the Number of Paths in a Grid Graph with Minimal Perfect Hash Functions," TCS Technical Report, TCS-TR-A-13-64, Apr. 2013
[3] M. Kholilurrohman, S. Minato, “Enumeration of graph walks visiting each vertex at most twice,” IEICE General Conference, DS-1-10, March, 2014.