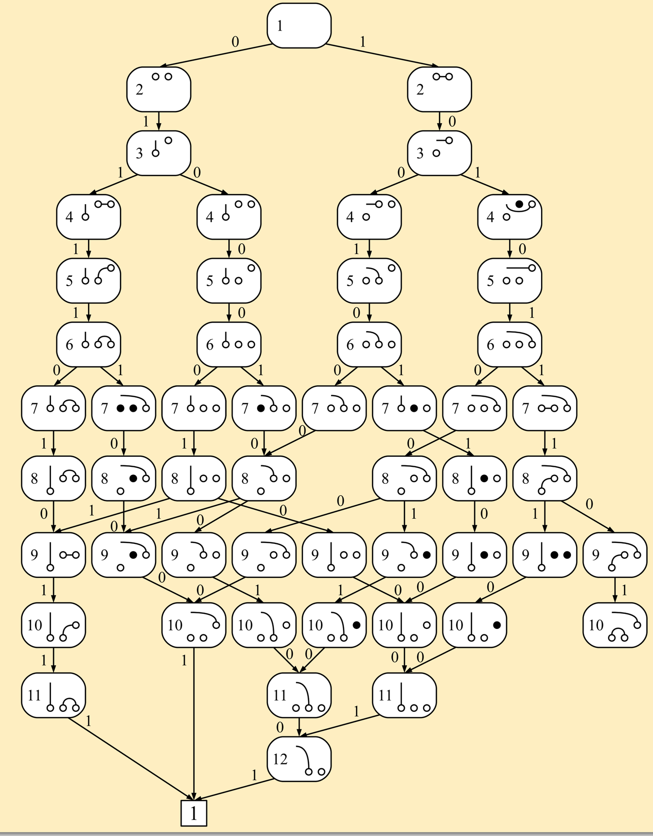
フロンティア法の応用に適したメモリ効率の良いZDD構築手法

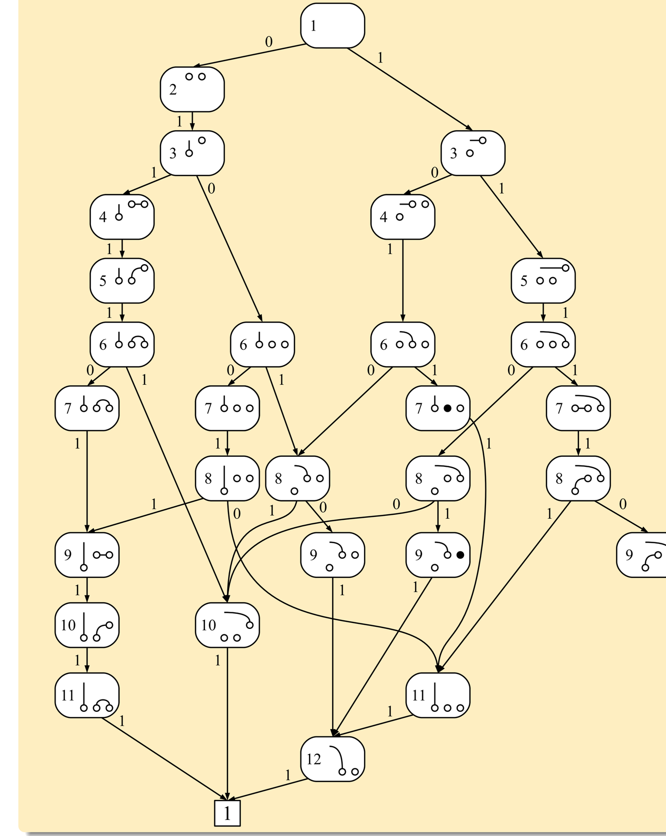
JST ERATO 湊離散構造処理系プロジェクト　岩下 洋哲

フロンティア法によるZDD構築



- 左の図は、3 × 3 グリッドグラフにおける v_1 から v_9 への全てのパスを表現するZDDを構築した様子。
- 初期状態から開始し、e_1 を使うかどうかで状態分岐、e_2 を使うかどうかで状態分岐...
- 同じ状態を一つの節点にまとめる。
- パスが完成したら1終端節点、パス完成の可能性が無くなったら0終端節点へ。

参考: ZDDの簡約化規則を考慮した改善手法

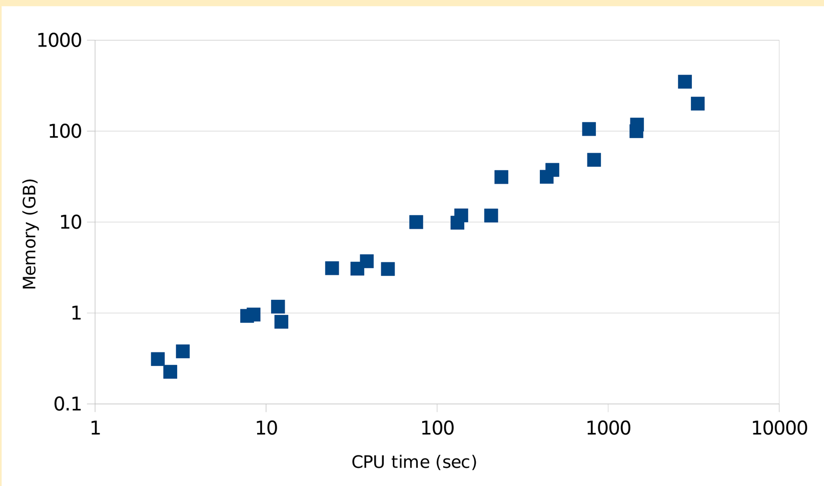


- 新しい状態を求めたとき、その1枝側の状態を「先読み」する。
- 1枝側が0終端節点であれば、さらに0枝側に状態を進める。

→ 冗長な節点をスキップ

課題

フロンティア法は高速だが、メモリを使い果たすのも早い。



≃ 100MB/sec

標準的なPCでは計算パワーに対してメモリが不足する傾向。

我々のアプローチ: 仮想ZDD

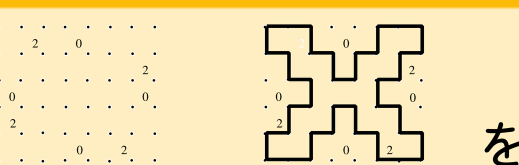
本当に必要になるまではZDD構造を作らないことにする。

- 必ずしも完成したZDD構造を持たない。
- 仮想ZDDオブジェクト P は次の機能を持つ:
  - $P.getRoot()$ ··· P の根節点を得る。
  - $P.getChild(p, 0)$ ··· 節点 p の0枝側の子節点を得る。
  - $P.getChild(p, 1)$ ··· 節点 p の1枝側の子節点を得る。
- 節点 p は変数レベルの情報 $p.level$ を持つ。
  （変数順序に対して降順、終端節点はレベル0）
- 仮想ZDDの形式でフロンティア法を実装しておく。

仮想ZDDの特徴

- それ自体は非常にコンパクト。
- 深さ優先／幅優先のZDD演算アルゴリズムがほとんどそのまま適用できる。
- 複数の仮想ZDDを結合して新たな仮想ZDDを作ることができる。

適用事例: リンクパズルソルバー

- スリザーリンクパズル  を次の２つの仮想ZDDの積と考えて解く。
  - P: 選択した辺集合がヒントとして与えられた数字の制約を満たす。
  - Q: 選択した辺集合が一つのサイクルを構成する。
- Q のZDD構造の構築はフロンティア法によるサイクル列挙そのもの。
- パズル本に掲載されている問題は 37 × 21 グリッドなど。
  → Q のZDD構造を主記憶上に構築することは不可能。
- 仮想ZDDを使えば Q のZDD構造を作らないで $P \cap Q$ を求めることが可能。

ZDD構造全体を構築することなく計算可能な性質は？

- より少ないメモリ使用量で計算したい。
- 例えば集合の要素数は上から下への幅優先探索で計算可能。
  - 根節点から各節点へのパス数をレベルごとに下へ伝播。
  - 合流したらパス数を加算。
  - １終端節点に集まった数が答え。
  - 処理済みのレベルの情報は捨てて良いので、メモリ量はZDDの幅でほぼ決まる。
  - 1TBマシンでのZDD構築は $G_{19,19}$ まで → カウントだけなら $G_{22,22}$ も可能（右表）。
- 最小コスト（最大コスト）も同様に計算可能。
- コスト最小（最大）の要素を出力することは ...

例: 仮想ZDD → ZDD構造

Construct(P, p)

1: **if** $p = 0$ **return** 0;
2: **if** $p = 1$ **return** 1;
3: **if** *opCache* has entry $p \mapsto r$ **return** r;
4: $i \leftarrow p.level$;
5: $r_0 \leftarrow Construct(P, P.getChild(p, 0))$;
6: $r_1 \leftarrow Construct(P, P.getChild(p, 1))$;
7: $r \leftarrow GetNode(i, r_0, r_1)$;
8: add $p \mapsto r$ to *opCache*;
9: return r.

例: 仮想ZDD ∪ 仮想ZDD → ZDD構造

ConstructUnion(P, p, Q, q)

1: **if** $p = 0$ **and** $q = 0$ **return** 0;
2: **if** $p = 1$ **or** $q = 1$ **return** 1;
3: **if** *opCache* has entry $\langle p, q \rangle \mapsto r$ **return** r;
4: **if** $p.level > q.level$ **then**
5: $i \leftarrow p.level$;
6: $r_0 \leftarrow ConstructUnion(P, P.getChild(p, 0), Q, q)$;
7: $r_1 \leftarrow P.getChild(p, 1)$;
8: **else if** $p.level < q.level$ **then**
9: $i \leftarrow q.level$;
10: $r_0 \leftarrow ConstructUnion(P, p, Q, Q.getChild(q, 0))$;
11: $r_1 \leftarrow Q.getChild(q, 1)$;
12: **else**
13: $i \leftarrow p.level$;
14: $r_0 \leftarrow ConstructUnion(P, P.getChild(p, 0), Q, Q.getChild(q, 0))$;
15: $r_1 \leftarrow ConstructUnion(P, P.getChild(p, 1), Q, Q.getChild(q, 1))$;
16: **end if**
17: $r \leftarrow GetNode(i, r_0, r_1)$;
18: add $\langle p, q \rangle \mapsto r$ to *opCache*;
19: return r.

例: 仮想ZDD ∪ 仮想ZDD → 仮想ZDD

$[P \cup Q].getRoot()$

1: **return** $\langle P.getRoot(), Q.getRoot() \rangle$.

$[P \cup Q].getChild(\langle p, q \rangle, c)$

1: $p' \leftarrow (p.level \geq q.level)\ ?\ P.getChild(p, c) : p$;
2: $q' \leftarrow (p.level \leq q.level)\ ?\ P.getChild(q, c) : q$;
3: **if** $p' = 0$ **and** $q' = 0$ **return** 0;
4: **if** $p' = 1$ **or** $q' = 1$ **return** 1;
5: **return** $\langle p', q' \rangle$.

$\langle p, q \rangle .level \equiv \max(p.level, q.level)$.

パス数の計算結果

	CPU (sec)	Mem (MB)	#paths	
$G_{2,2}$	0.0	1	2.00E+000	2
$G_{3,3}$	0.0	1	1.20E+001	12
$G_{4,4}$	0.0	1	1.84E+002	184
$G_{5,5}$	0.0	1	8.51E+003	8512
$G_{6,6}$	0.0	1	1.26E+006	1262816
$G_{7,7}$	0.0	1	5.76E+008	575780564
$G_{8,8}$	0.0	2	7.89E+011	789360053252
$G_{9,9}$	0.0	4	3.27E+015	3266598486981642
$G_{10,10}$	0.1	4	4.10E+019	41044208702632496804
$G_{11,11}$	0.2	8	1.57E+024	1568758030464750013214100
$G_{12,12}$	0.8	20	1.82E+029	182413291514248049241470885236
$G_{13,13}$	3.4	43	6.45E+034	64528039343270018963357185158482118
$G_{14,14}$	13.8	124	6.95E+040	69450664761521361664274701548907358996488
$G_{15,15}$	51.0	296	2.27E+047	227449714676812739631826459327989863387613323440
$G_{16,16}$	177.7	788	2.27E+054	2266745568862672746374567396713098934866324885408319028
$G_{17,17}$	606.6	2,270	6.87E+061	68745445609149931587631563132489232824587945968099457285419306
$G_{18,18}$	2012.9	6,835	6.34E+069	6344814611237963971310297540795524400449443986866480693646369387855336
$G_{19,19}$	6534.8	18,575	1.78E+078	1782112840842065129893384946652325275167838065704767655931452474605826692782532
$G_{20,20}$	23263.0	56,155	1.52E+087	1523344971704879993080742810319229690899454255323294555776029866737355060592877569255844
$G_{21,21}$	81139.8	167,137	3.96E+096	3962892199823037560207299517133362502106339705739463771515237113377010682364035706704472064940398
$G_{22,22}$	266482.1	499,839	3.14E+106	31374751050137102720420538137382214513103312193698723653061351991346433379389385793965576992246021316463868