アルゴリズム特論（第６回）

北海道大学　大学院　情報科学研究科
アルゴリズム研究室　　湊　真一

前回の内容

- **BDD処理系の実装技術**
  - BDDデータ構造の復習
  - データ構造
    - 節点の表現、テーブル上の表現
  - 節テーブルによる一意性の保証
    - ハッシュテーブルの実装
  - 論理演算アルゴリズム
    - 再帰的アルゴリズム、シャノンの展開
    - 演算例
    - 演算キャッシュの実装と計算時間
    - 否定演算と否定エッジ
    - 代入演算と計算時間
  - 記憶管理
    - 参照カウンタとガベジコレクション、演算キャッシュの初期化
    - 記憶領域の動的拡張

今回の内容

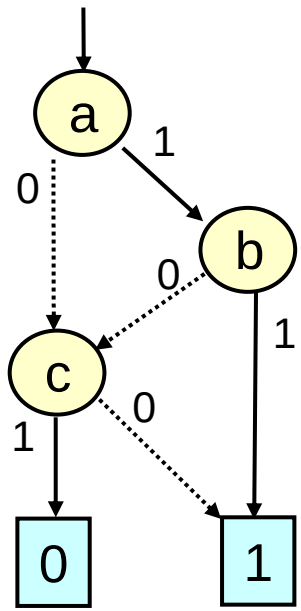
- **BDDの変数順序づけ**
  - 順序づけの影響の例
  - 2つの経験則
    - 論理式・論理回路から生成する場合の重み付け法
  - 与えられたBDDの最小化
    - 厳密解法：全解列挙、動的計画法（効果と計算時間）
    - 近似最適化法：隣接交換法、最小幅法（効果と計算時間）
  - 動的順序付け法
    - BDD処理系への組み込み
    - 2変数の入れ替えの局所性
    - 節テーブルハッシュの横割り方式
    - 効果と計算時間

BDDの変数順序づけ

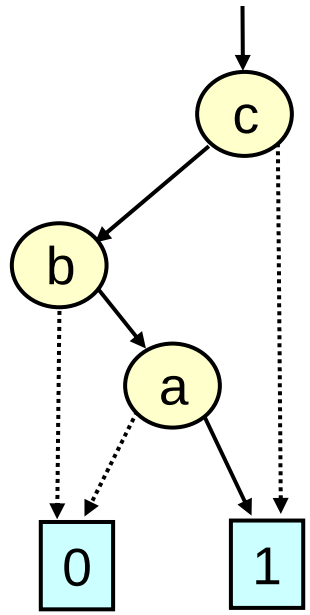
- BDDは変数の順序を固定すれば、論理関数に対して形が一意に決まる
- 変数順序が変わると、同じ論理関数で異なる形となることが多い。節点数も変化する。
- 順序づけの影響の大きさは論理関数の性質に依存
  - 対称関数では全く変化しない
  - 数百倍もの差が生じる場合もある
- 良い順序づけを見つけることは実用上重要な問題
  - 論理式からBDDを作る場合に、変数順が悪いと、節点数が爆発的に増大し、記憶あふれを起こして計算結果が得られないことも

変数順序によるBDDの変化

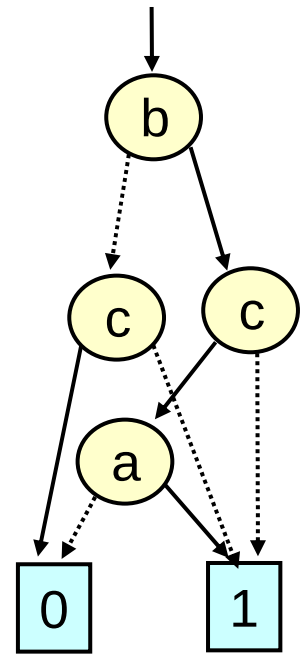
F = a b + ~c



F = a b + ~c



F = a b + ~c



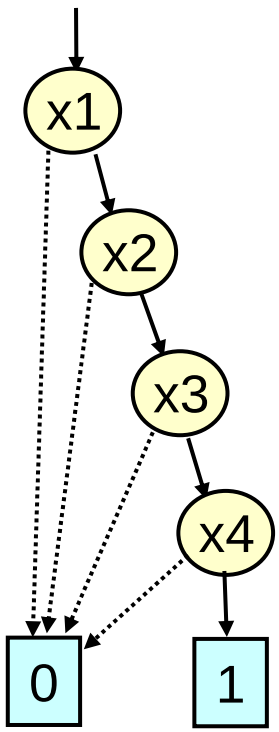
変数順序による影響パタン

(a) どんな順序でも簡単に表せる論理関数
- 論理積・和・パリティ関数
- 対称関数（多数決論理など）

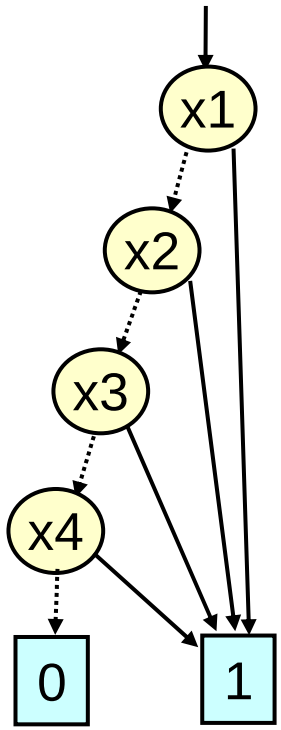
(b) 良い順序のときだけ簡単に表せる論理関数
- AND-ORの木状の論理回路（アキレス腱関数など）
- 算術加減算、比較
- データセレクタ

(c) どんな順序でも簡単に表せない論理関数
- 算術乗算・除算
- ランダム関数

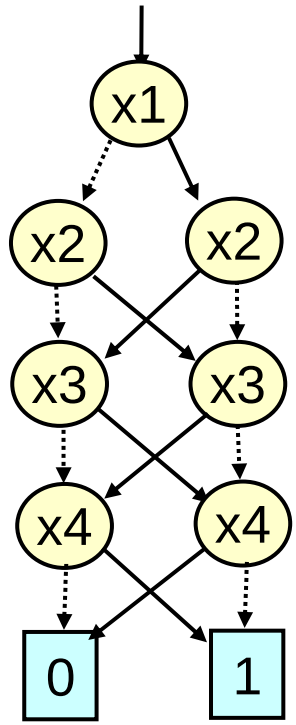
n変数の論理積・論理和・パリティ



論理積(AND)



論理和(OR)

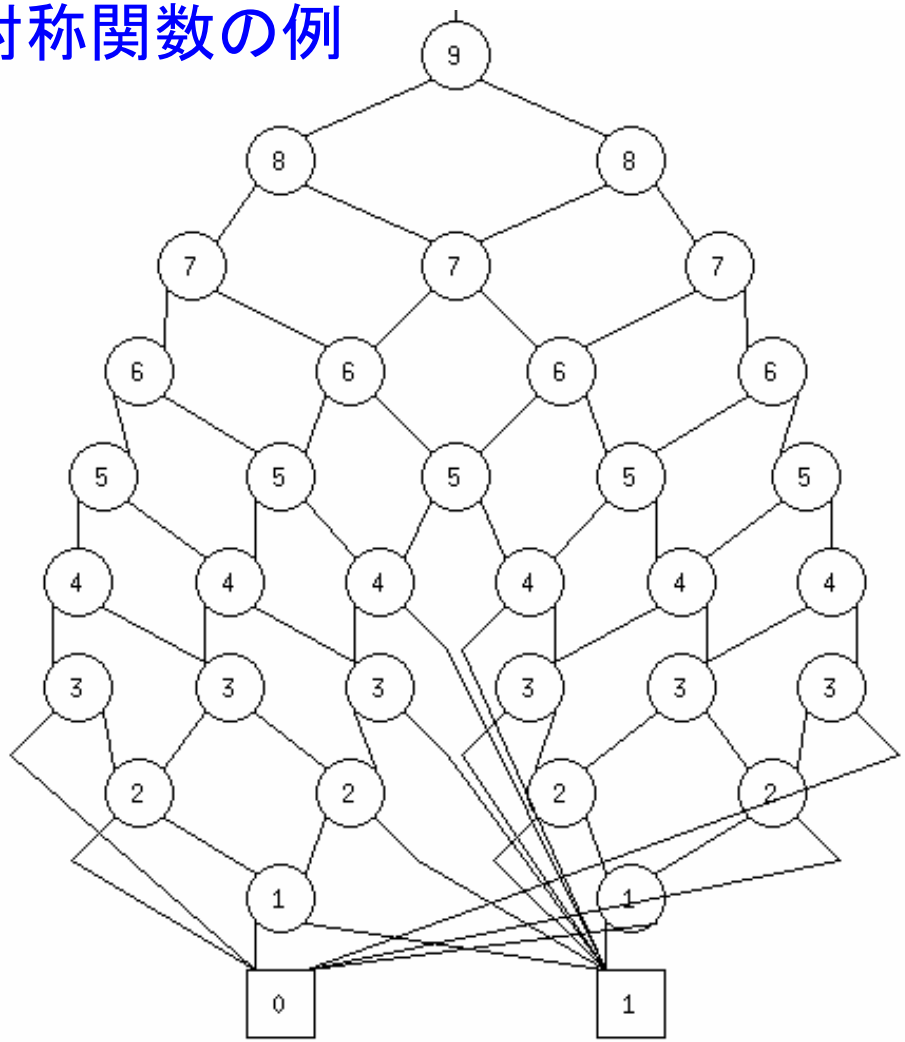


排他的論理和・
パリティ(EXOR)

- いずれも、変数順序に関わらず nに比例する節点数 O(n) で表現可能

対称(symmetric)論理関数

9入力変数の
対称関数の例

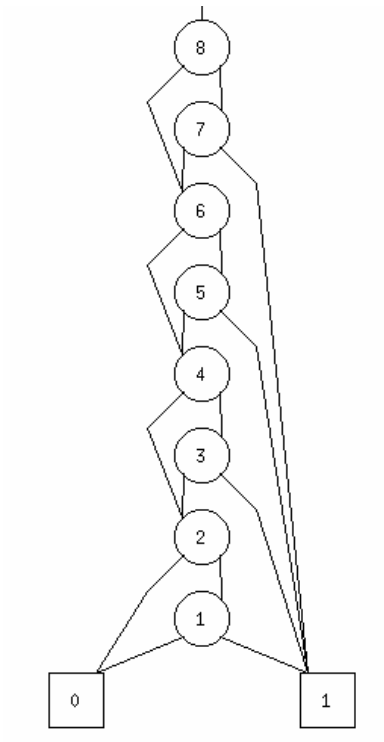


- 対称関数：変数の順序を入れ替えても論理に影響がない論理関数
  - n個の入力変数のうち、”1”になっている変数の個数によって出力の値が決まる
- BDDの場合、各段ごとに、最上位からその変数までの”1”の個数を示している。
  - BDDの幅は最大でn
  - 全体の節点数は$O(n^2)$

順序づけの影響が顕著な例(1)

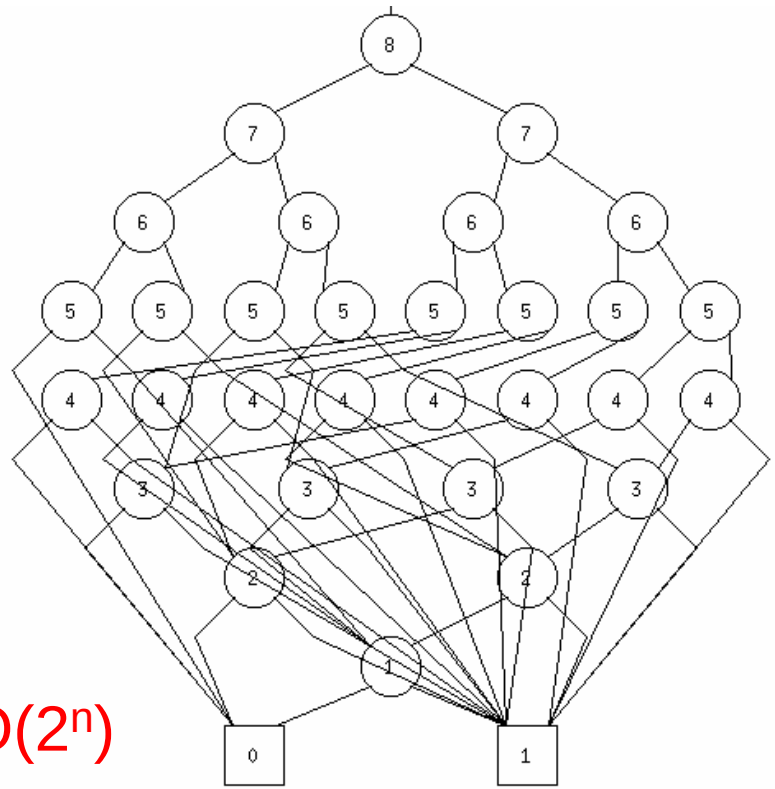
- BDDは変数の順序づけにより節点数が著しく変化する場合がある。

x1 x2 + x3 x4 + x5 x6 + x7 x8



O(n)

x1 x5 + x2 x6 + x3 x7 + x4 x8

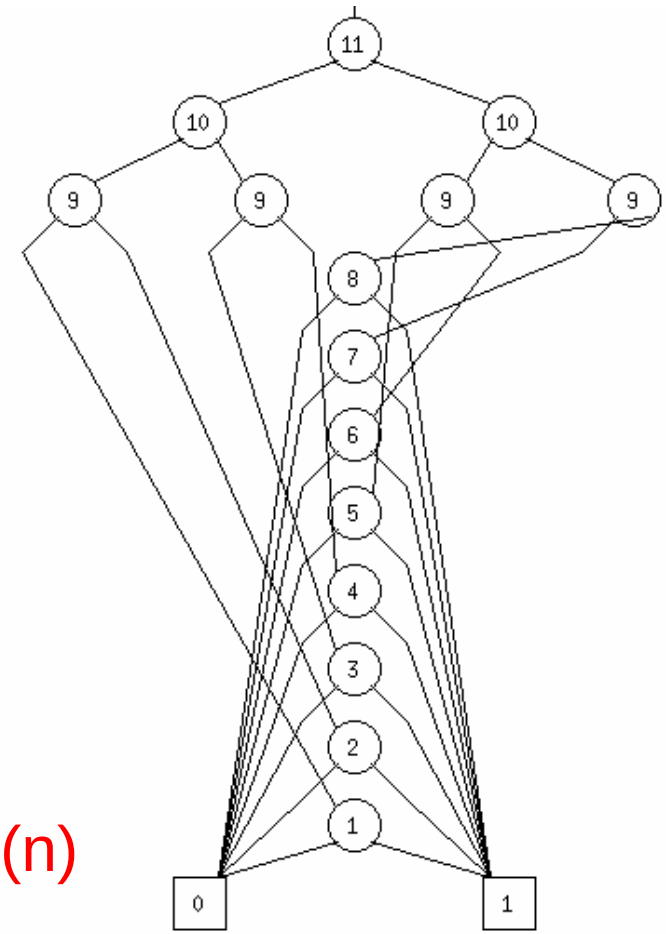


O(2^n)

順序づけの影響が顕著な例(2)

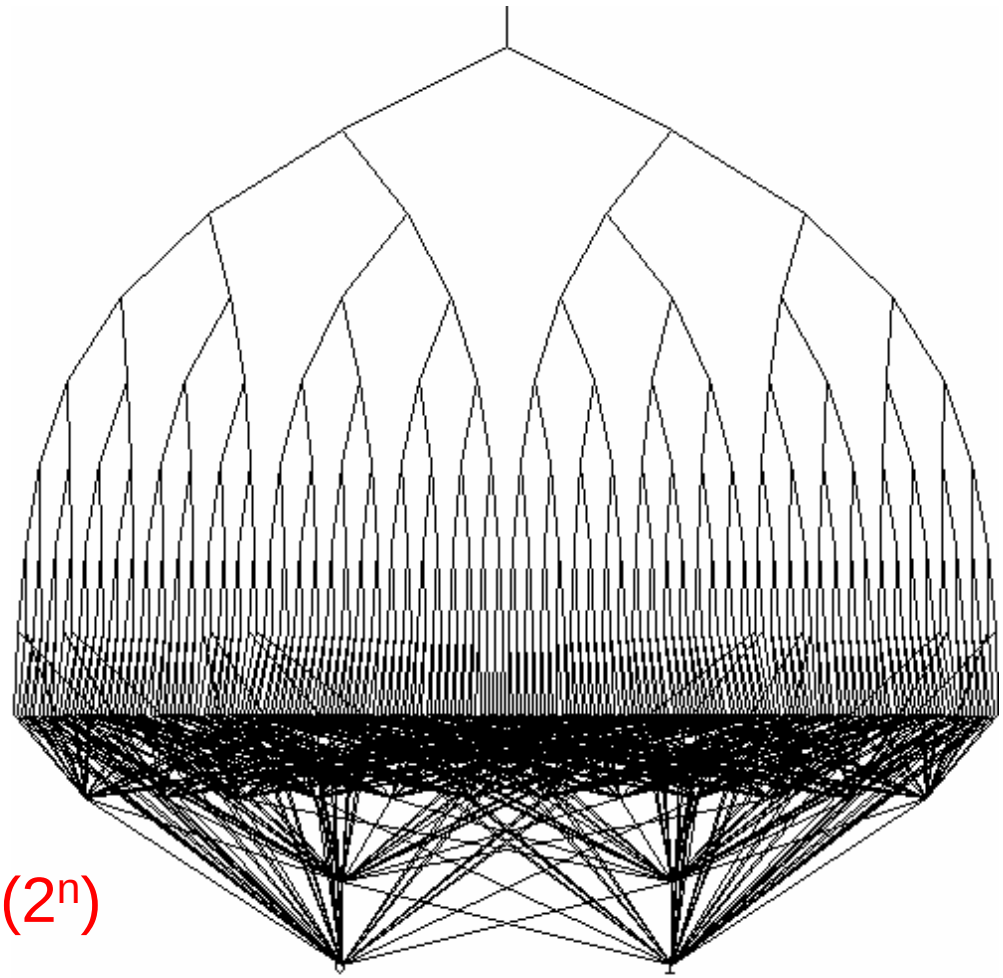
- 8入力データセレクタ

制御入力を上位にした場合



O(n)

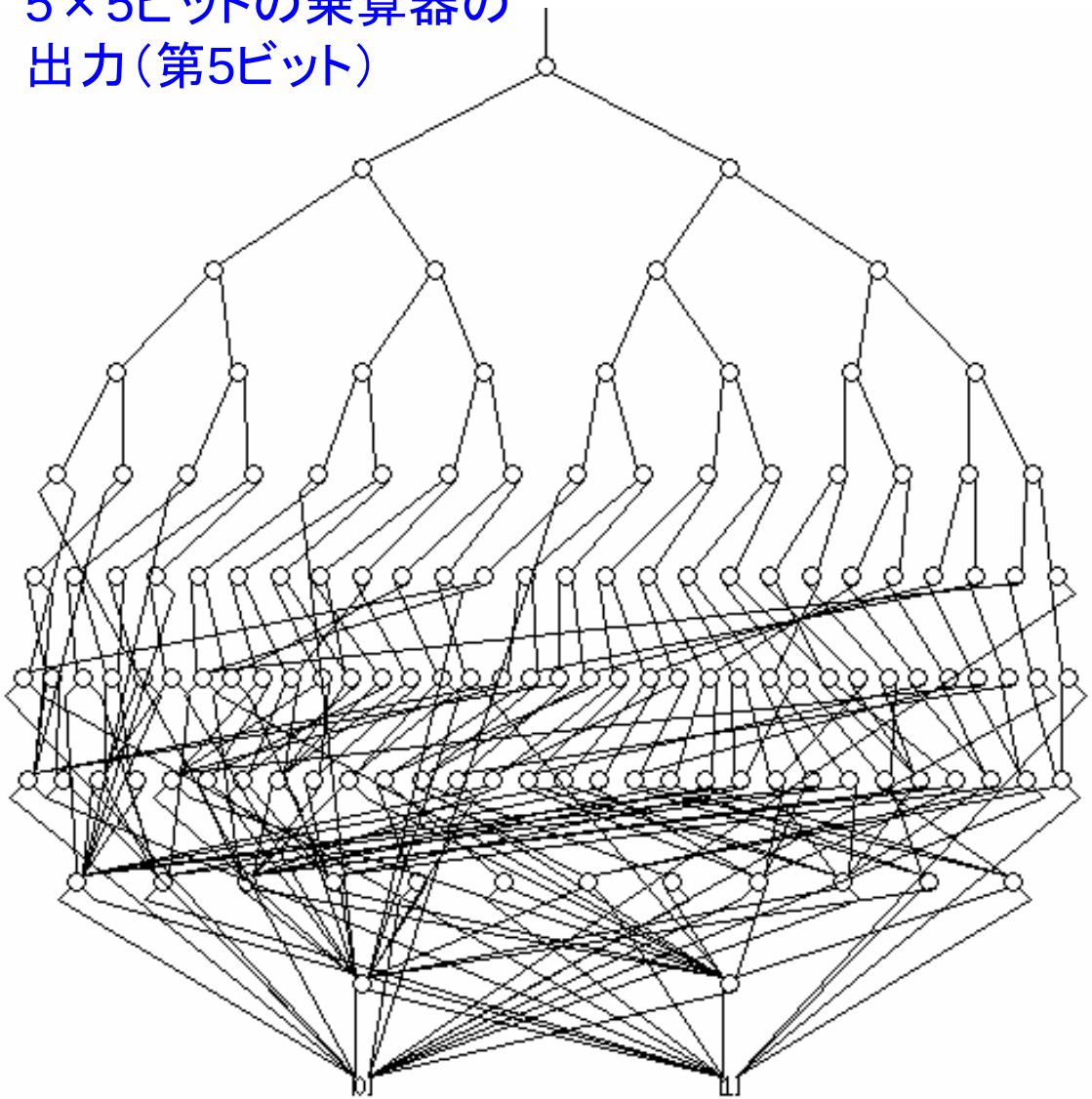
データを上位にした場合



O(2^n)

順序づけによらず複雑になる例

5×5ビットの乗算器の
出力（第5ビット）



- 2進数の算術乗算は苦手
  - どんな変数順序でも、$O(2^n)$の節点数になることが数学的に証明されている。
  - 除算も同様に常に指数オーダとなる

2つの経験則

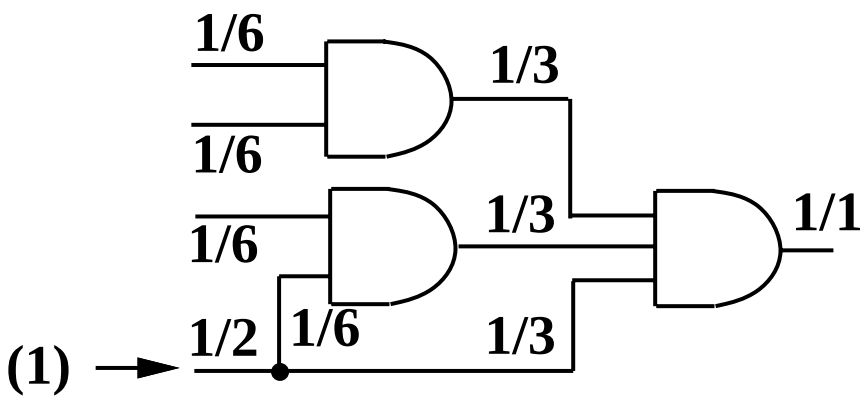
- 局所計算性:
  「互いに関係の深い変数は近い順序にした方が良い」
  - AND-OR2段論理式の例
  - 算術加算・比較
- 出力制御性:
  「出力を制御する力の強い変数は上位にした方が良い」
  - データセレクタの例

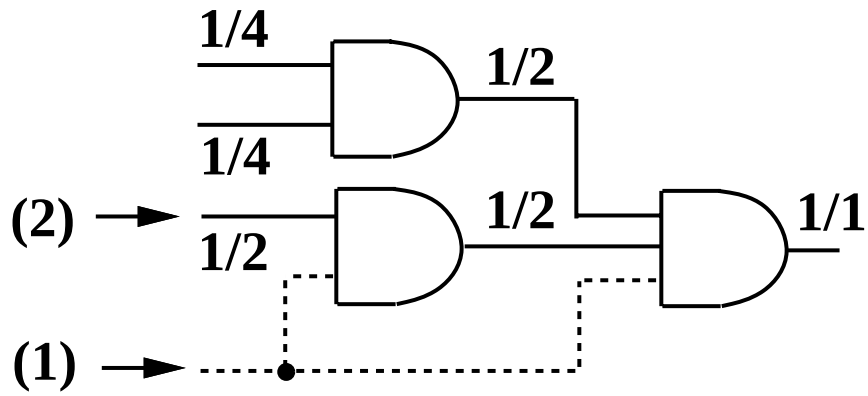
2つの経験則が互いに衝突する場合もある。
一般の例題では両者を満たすのは難しいことが多い

論理式や回路の構造を用いた順序づけ法

- ２つの経験則を人手で適用するのではなく、
  論理式や回路の構造を機械的にたどって順序づけを行う、発見的手法がいくつか提案されている。
  - 深さ優先探索法[Fujita他1988]
    論理式（または回路）の出力側から深さ優先順にたどっていき、先に到達した入力の変数から上位に並べる
  - 動的重みづけ法[Minato他1989]
    論理式（または回路）の出力側から重みを入力側に分配しながら伝えて行き、重みが最も大きい入力の変数を選び最上位にする。選んだ入力に関係する線を取り除き、再度重みを計算し、次の変数を選ぶ。以上を繰り返す。

動的重みづけ法の計算例





出力を制御する力の強い
入力変数に重みが多く
配分される

→ 制御性の強い変数が
上位になりやすい

選択した入力変数と関係が
強い入力変数の重みが増加

→ 関連する変数が
近い順序になりやすい

動的重みづけ法の実験結果

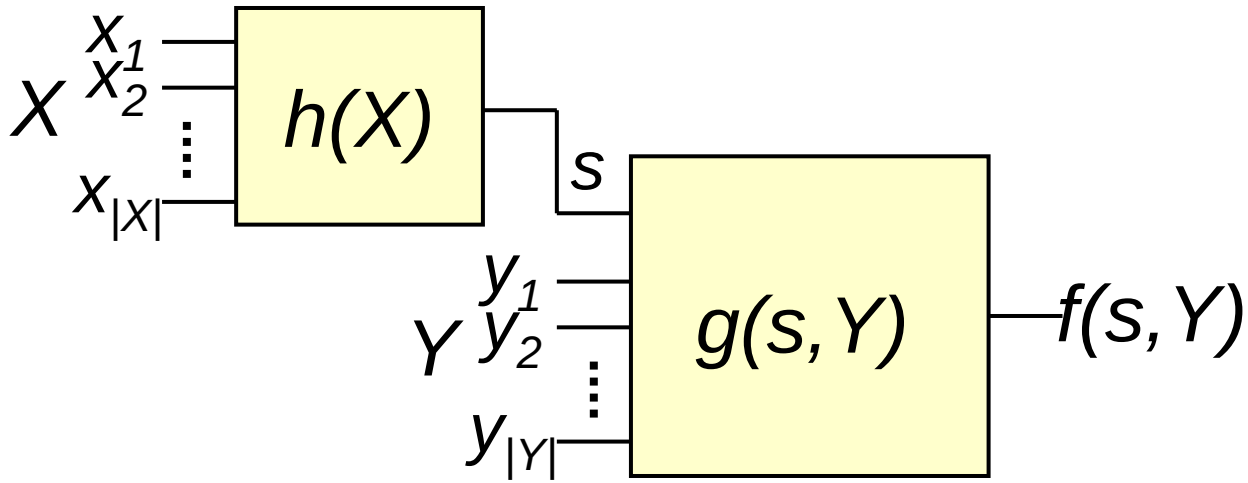
Circuit	(A) BDD nodes	(A) Time (sec)	(B) BDD nodes	(B) Time (sec)	(C) BDD nodes	(C) Time (sec)	(D) BDD nodes	(D) Time (sec)
sel8	23	0.1	16	0.1	510	0.1	88	0.1
enc8	28	0.1	28	0.1	23	0.1	29	0.1
add8	41	0.2	83	0.1	222	0.2	885	0.2
add16	81	0.3	171	0.2	830	0.4	19224	1.4
mult4	150	0.1	139	0.1	145	0.1	153	0.2
mult8	10766	2.7	9257	2.4	9083	2.2	15526	4.1
c432	27302	6.1	3987	1.2	1732	0.7	(>500k)	-
c499	52369	7.2	115654	11.6	45921	7.7	(>500k)	-
c880	23364	2.1	(>500k)	-	(>500k)	-	(>500k)	-
c1355	52369	8.1	115654	15.4	45921	8.4	(>500k)	-
c1908	17129	5.1	23258	4.8	36006	9.7	77989	23.5
c5315	31229	9.4	57584	8.1	(>500k)	-	(>500k)	-

(A): Using dynamic weight assignment method, (B): In the original order,
(C): In the original order (reverse), (D): In a random order

計算時間は（順序づけ処理＋BDD生成）の時間 （SUN SPARC1)

ほとんどの例で [順序づけ＜元のまま＜乱数順] （一部例外あり）

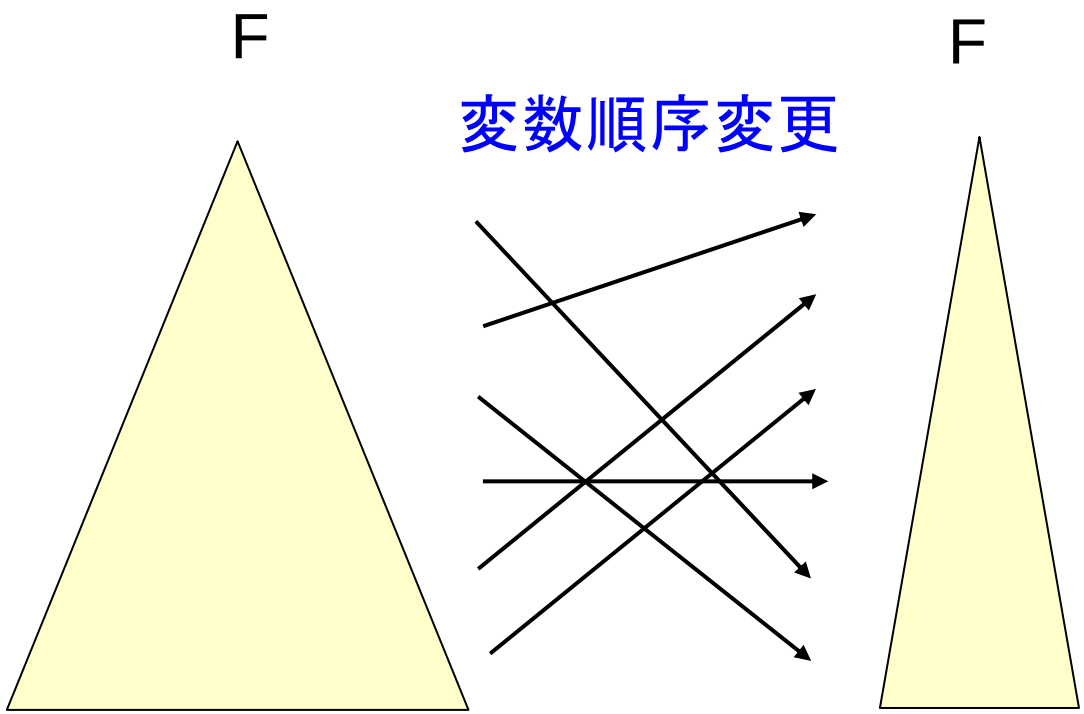
構造を用いた順序づけ法がなぜ良いのか



- 論理式や回路を上記のように切り分けることができる場合は、XとYをグループ化して順序づけするとBDDが小さくなることが、数学的に示されている。
  - 実際には枝分れや再収れんがあるので、きれいに書けないことが多い
- 論理式や回路の構造をもとにした順序づけ方法（深さ優先探索など）は、上記のような変数順を自動的に見つけ出している。
  - 人が見やすいように書いた図の変数順が意外と良い

与えられたBDDの最小化問題

- ある適当な順序づけのBDDを与えられたときに、節点数を最小にする変数順序を求める問題
  - 任意の論理関数について最適順を求める問題はNP完全であることが示されている。[Tani他1993]
  - ただし、元のBDDが小さければ20～30変数でも最適解を求められる場合がある

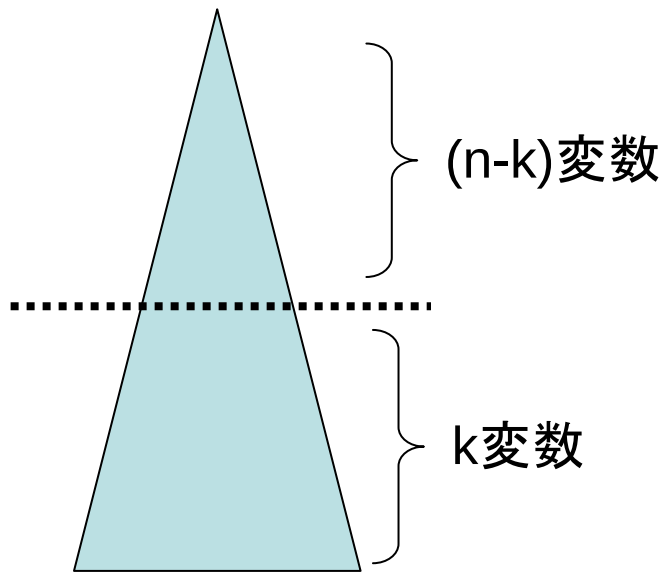


しらみつぶし法による最小化

- 一番わかりやすいアルゴリズム
  - n 変数について、全ての順列（n!通り）を作り、それぞれの順序づけのBDDを全て作る。その中で節点数が最小のBDDが最適順序。
  - 計算時間はO(n! |G|)
    （ただし|G|は全ての順序のBDDの平均節点数。最悪$2^n/n$）
  - とても実用的ではない（n=8,9くらいでギブアップ）

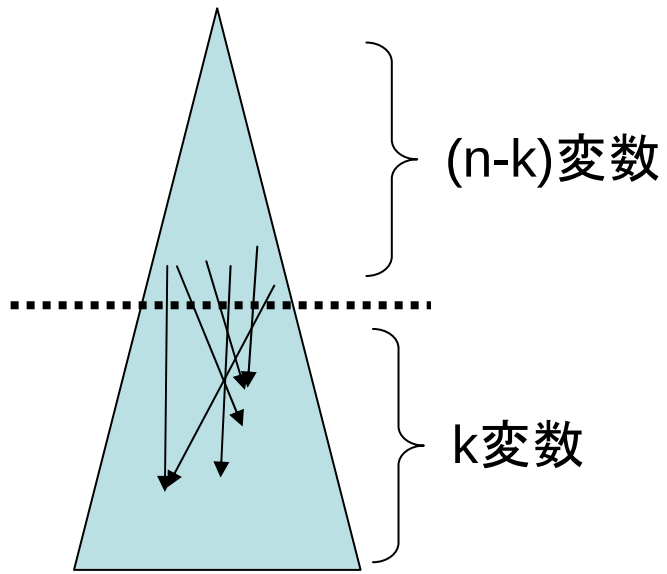
動的計画法を用いた高速化

- 全部の順列をしらみつぶしで試すのは無駄が多い
- 以下の性質を利用して高速化 [Freedman他1987]
  - BDDをある変数を境に上位/下位にグループ分けしたとき、グループ内の変数の順序を変えてもグループ外の変数の節点数に影響しない
- アルゴリズムの概略:
  - n変数の中からk個の部分集合を選びBDDの下位に並べるときの最適解を、メモリに記憶しておく。($_nC_k$通りを全て計算して記憶する)
  - 下位k変数の最適解を使って、下位(k+1)変数の全ての組合せの最適解を求めて記憶する。
- 計算時間: $O(n^2 3^n)$
  - しらみつぶしより相当よいが、まだたいへん



探索空間の枝刈りによる高速化

- さらに、変数の順序づけをしている最中に、ある程度以上BDDが大きくなったら、その部分集合の組合せは調べても無駄なので探索を打ち切ってよい
- グループ境界を横切る枝の行き先の数をMとすると、どんなに順序づけをがんばっても、下位グループの節点数はMより少なくできない
  - Mが大きくなりすぎたらやめて別の組合せを探す
  - 初期BDDが小さければ高速化効果が大きい
  - 厳密な計算量は不明
    BDD節点数が100～1000まで
    20～30変数までならほぼ1日以内に
    厳密解を求められる[Drechsler他1998]

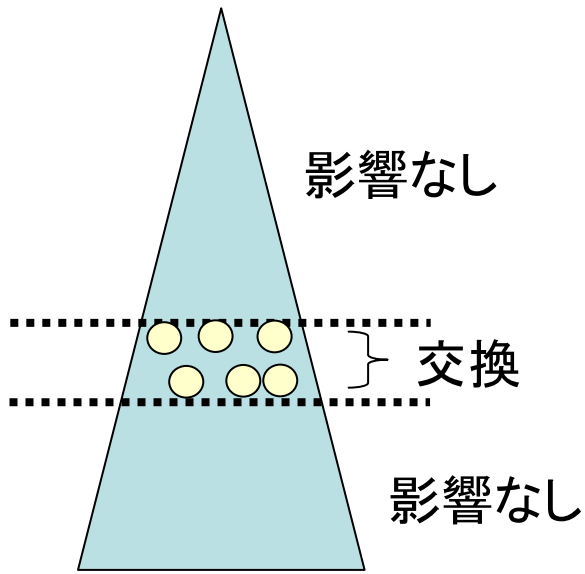


与えられたBDDの変数順序改良問題

- 少し大きな規模のBDDを最小化するのは、計算時間がかかりすぎて困難
- 実用的には、少しでも節点数を減らして、メモリを節約し、計算時間を短縮したい
  - 初期BDDは適当な変数順序で何とか生成できたとする
  - 変数順序を入れ替えて、節点数を削減させる
  - 限られた時間内になるべく良い結果を得たい
- 大規模なBDDでは最適解が知られていないので、ベンチマーク問題を用いた実験結果で評価される

隣接変数交換による節点の変化

- 隣接する2変数の順序交換は比較的高速に実行可能
  - 2変数以外に関する節点は影響しない
  - 計算時間は関係する節点の数に比例
    （つまり論理演算1回分くらいの計算時間）
- 隣接変数の交換を繰り返せば
  任意の変数順序づけが可能
  - ある1つの変数を下から上まで
    移動させることもできる
  - 変数を大きく移動させると
    全体の節点数が大きく増える
    場合もある



バブルソート型の順序入替え法

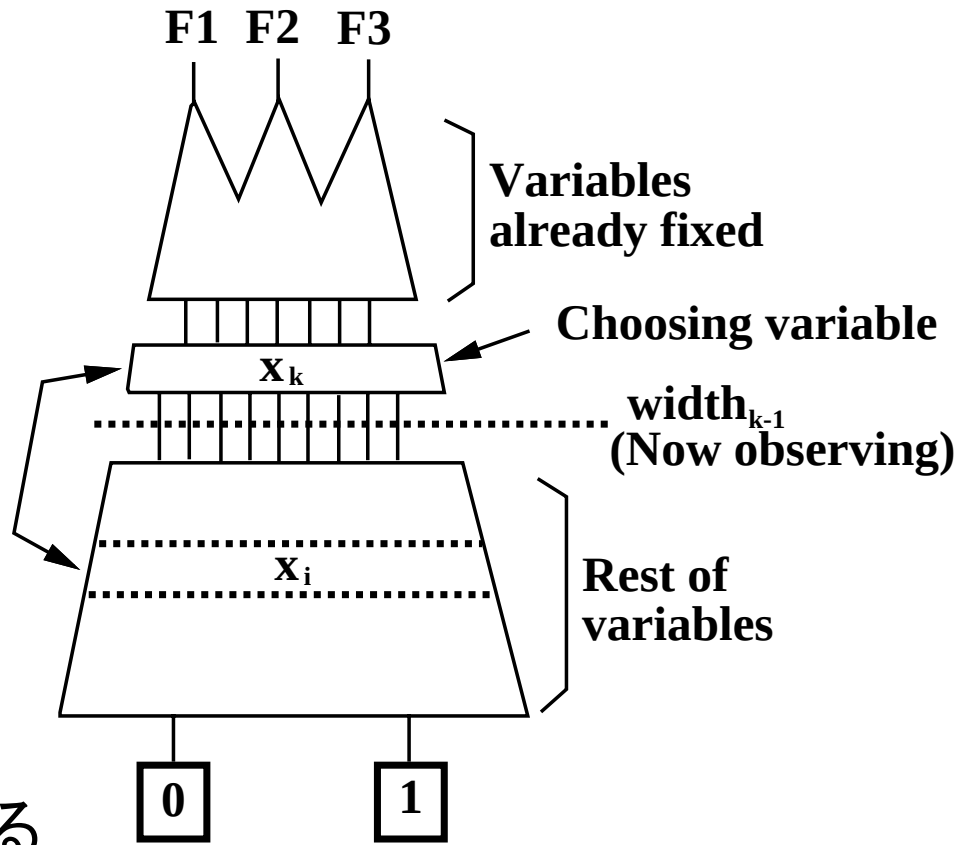
- 隣接変数を交換して節点数が減れば採用し、減らなければ元に戻す
  - 貪欲法 (Greedy method)
  - どの隣接変数をとっても改善しなくなるまで繰り返す。
    （時間に応じて途中でやめてもよい）
- 長所:
  - 途中状態を含めて初期BDDより悪くなることはない
  - 比較的高速
- 短所:
  - 局所最適解につかまりやすい
    （もっと良い順序づけがあるのに抜け出せなくなる）
  - 初期BDDの順序に大きく依存する

Sifting（ふるいがけ）順序づけ法

- ある1つの変数に着目し、隣接変数交換を繰り返しながら、最上位～最下位の全ての順位を経験させる
  - 他の変数の相対的順位は変えない
- 全ての順位の中で最も節点数が少なかった順位に移動させ、そこで固定する。
- また別の変数について同じことを繰り返す。
  - 全ての変数についてそれ以上改善しなくなったら終わり（時間に応じて途中でやめてもよい）
- 最上位から最下位まで動かす間に、BDDが著しく増大する場合がある。
  - 元のサイズの2倍を超えたらそれ以上遠くには動かさないという発見的手法も有効
- 長所：元のBDDより悪くはならない
  局所最適解につかまりにくい
- 短所：計算時間は比較的長い（変数交換回数がn^2以上）

BDDの「幅」に着目した順序づけ法

- 上下のグループ境界を横切る枝の行き先の数をBDDの「幅」と呼ぶ
  - 下位グループの節点数の下界となる
  - 幅を大きくするような変数を上位に持ってくるのは不利
- BDDの最上位から順にグループ境界の幅を最小にするような変数を選んで固定していく
- 長所：
  - 局所最適解に捕まりにくい
  - 初期BDDに依存しにくい
- 短所：
  - 計算時間は比較的長い
  - 元のBDDより悪くなる場合もある



順序づけの効果と処理時間

- BDDの幅を用いた順序づけ法の実験結果

Function	In.	Out.	BDD nodes Before	BDD nodes After	Time (sec)
c432	36	7	27302	1361	188.3
c499	41	32	52369	40288	1763.6
c880	60	26	23369	9114	874.6
c1908	33	25	17129	8100	252.3
c5315	178	123	31229	2720	6389.3

（計算機：SAPRC 2, 32MB）

BDD節点数 1万～10万
入力変数の個数　30~200くらい
計算時間：数時間以内
効果：20%削減～20分の1（元の順序の良し悪しに依存）

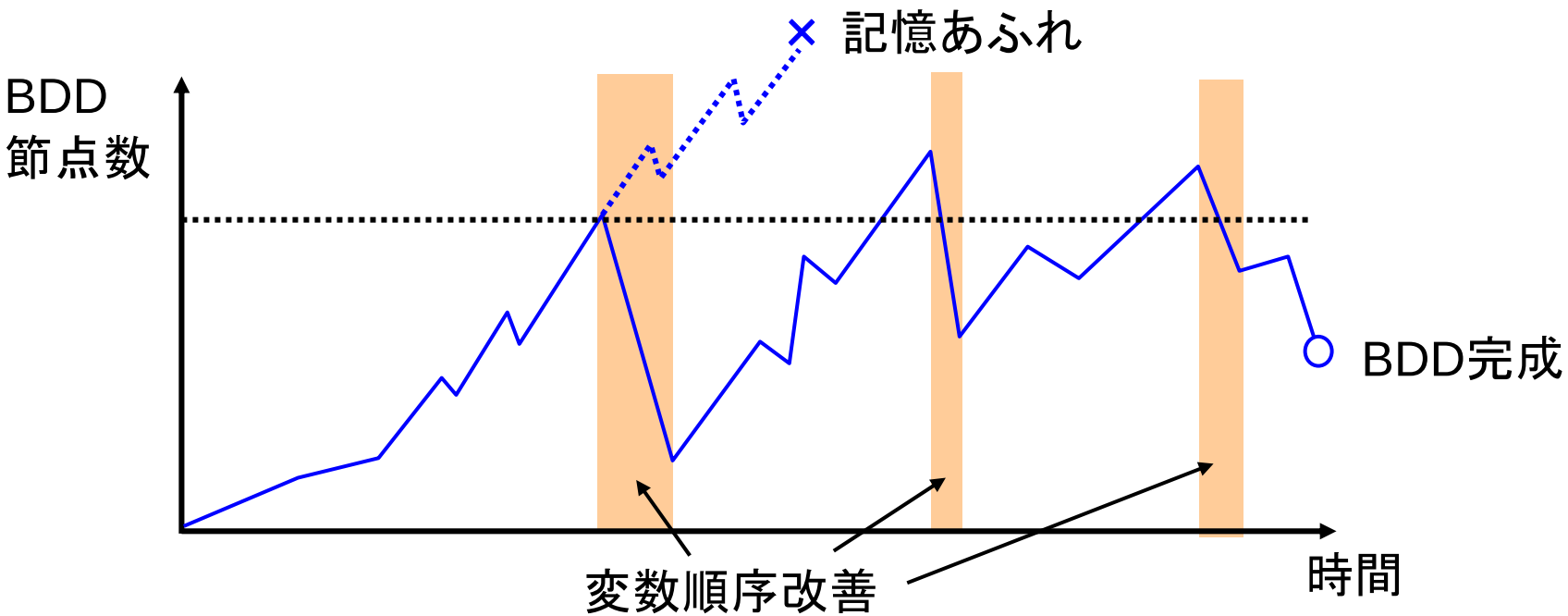
動的順序づけ法

- 与えられたBDDの変数順序を改善する方法では、初期BDDをどうやって作るかという問題がある
  - 初期順序が悪いと節点数が爆発的に増大して、初期BDDを作れない
  - BDDを作る前に比較的良い変数順序をみつける必要がある
- 論理式や回路の構造を利用する方法は、多くの場合、比較的良い順序をみつけることができるが、発見的手法なのではずれることもある

BDDを作りながら変数順序を改善する
動的順序づけ法(Dynamic reordering)という技法

動的順序づけ法のしくみ

- BDDの論理演算の最中に、処理系全体の節点数がある限界を超えたときに、自動的に変数順序入替えプログラムが走り、節点数を削減してから、演算処理を再開する
  - 不要節点のガベジコレクションと同様の自動化処理



動的順序づけ法の効果と処理時間

- 動的順序づけ法により、今までにBDDを構築することができなかった大規模な問題（の一部）が構築できるようになった（BDDの適用範囲が広がった）。
- 動的順序づけ法は、処理系が勝手に走らせてくれるので、ユーザは変数順序のことをあまり気にしなくてもよくなった
  - ただし、良い順序がわかっていれば、最初からその順序を指定したほうが効率的である。
- 動的順序づけ法は効果的だが、計算時間はかなりかかる。
  - 今までは数分後に記憶あふれであきらめていた問題を、順序づけをしながら、ゴリゴリと数十時間かけて計算する。
  - さんざん時間をかけて計算しても、最後にどうしても記憶あふれする場合もある。（やってみないとわからない）

BDD変数順序づけ法のまとめ

- BDDの変数順序づけアルゴリズムは1990年代に盛んに研究された。
  - BDDを作る前に論理式や回路の構造を用いて順序づけを行う発見的手法
  - 与えられたBDDを最小にする順序を求めるアルゴリズム
  - 与えられたBDDに対して、限られた時間で比較的良い順序を求めるアルゴリズム
  - BDDを作りながら、自動的に変数順序改善を行う技法
- 必要に応じて、上記の手法を組合せて用いると良い