1D5-OS-02b-5

重み付き二分決定グラフを用いた組合せ最適化手法

Combinatorial Optimization with Weighted Binary Decision Diagrams

岩下 洋哲
Hiroaki Iwashita

株式会社富士通研究所
Fujitsu Laboratories Ltd.

A new combinatorial optimization method based on weighted binary decision diagrams (WDDs) is presented. It finds one or all 0-1 input vectors minimizing the total output values of given multiple WDDs. An obvious method is to build a monolithic WDD by summing the WDDs before searching solutions; however, it often leads to a WDD blowup. Another possible method is to simultaneously trace the multiple WDDs as they are; it often spends much CPU time because search-space pruning does not work very well. Our method transforms a set of WDDs for earlier search-space pruning, by applying additive decomposition and additive composition repeatedly to them while avoiding a WDD blowup.

1. はじめに

大規模な組合せ集合を表現するデータ構造として二分決定グラフ (BDD: binary decision diagram) [Bryant 86, Minato 93] が利用されている．近年ではグラフに含まれる木や経路などの様々な構造を列挙できるフロンティア法が開発され，組合せ集合表現としての BDD の応用範囲がさらに広がった [Kawahara 14, Inoue 14a, Iwashita 13, ERAT 14]．これを利用すると，例えば実用規模の配電網における 468 個のスイッチの開閉条件を BDD で正確に表現し，その中で電力の無駄を最小化するスイッチ開閉パターンを求めることも可能になる [Inoue 14b].

BDD は膨大な組合せをコンパクトに表現するが，実際の場面ではその中から目的に合ったものを取り出したいことが多い．それらは，制約条件が BDD で与えられた組合せ最適化問題である．特に，入力変数 $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ に対して目的関数が入力変数に関して線形，すなわち $f(\mathbf{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + c$ の形である場合には，実数係数 w_i をレベル i の BDD 節点から出る 1-枝の重みと考えることにより，BDD 上の最短経路問題として（節点数に比例するコストで）解くことができる [柳谷 93]．また，目的関数が有向非巡回グラフ (DAG: directed acycric graph) 上の経路長に対応する場合に，BDD で与えられた制約条件と DAG とを同時に探索することで最適解を得る手法も提案されている [Nishino 15].

これらの方法は，制約条件が単一の BDD で表現されることを前提としている．ところが現実の問題では，複数の（互いに異なった性質を持つ）制約条件を考慮しなければならないことが多く，一見して単純な制約条件の集まりであっても一つに合成しようとすると BDD サイズが爆発してしまうケースが少なくない．合成を避けて制約条件ごとの BDD を同時に探索しても良いが，その場合には探索の早い枝刈りができず，結果として現実的な時間内で最適解が求まらないケースが多くなる．

例えば多次元ナップサック問題は，一つの目的関数と複数の制約条件から成る．目的関数は線形式なので，変数と同じ数の非終端節点を使った WDD で表現できる．また，一つ一つの制約条件はしきい値関数であり，一般に BDD サイズの爆発を起こしにくい [Knuth 11]．しかしながら，制約条件の数が多い場合にそれらを一つの BDD に合成することは，必ずしも容易ではない [Behle 07].

連絡先: 岩下 洋哲，iwashita.hiroak@jp.fujitsu.com

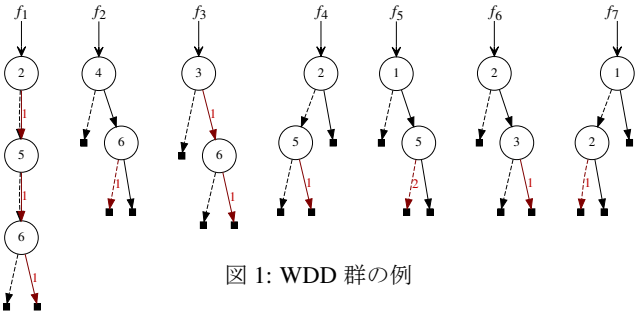

図 1: WDD 群の例

重み付き Max-SAT 問題の場合は，節ごとにそれぞれが充足されない場合に発生する重み（ハード節は無限大の重み）を表現した関数を考えると，それらの総和が目的関数に相当する．それぞれの関数は節に現れるリテラルと同じ数の非終端節点を使って容易に WDD で表現可能であるが，それらを単一の WDD に合成することは All-SAT の解を単一の BDD として求める問題 [Toda 15] と同等以上に難しい．一方で，合成を避けて節ごとの WDD を WDD を同時に探索するアルゴリズムは，変数順序を固定した二分探索と状態のキャッシングだけで Max-SAT 問題を解いているようなものである．

我々は，重み付き二分決定グラフ (WDD: weighted BDD) を用いた枠組みの中で問題を考える．最大化問題と最小化問題は目的関数の符号付けで相互に変換できるため，ここでは議論を最小化問題に絞る．最小化問題における制約条件は，無限大の制約違反コストを持った関数として WDD で表現することができる．制約条件が既に BDD で表現されている場合には，0-終端に接続する枝の重みを無限大，それ以外の枝の重みを 0 とすればそのような WDD に変換できる．そこで本文では，目的関数または制約条件を表す関数群 $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})$ が WDD のデータ構造で与えられることを前提とし，それらの和 $f(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x}) + \cdots + f_m(\mathbf{x})$ を厳密に最小化する値割り当ての一つまたは全てを求める問題について考える．

2. 重み付き二分決定グラフ

WDD の例を図 1 に示す．WDD は 0-1 ベクトルを入力として実数 [*1] を出力する関数を表現する．一般的な BDD と同様

*1 ここでは制約条件の取り扱いを単純化するため正の無限大（∞）を加えた拡張実数を考える．

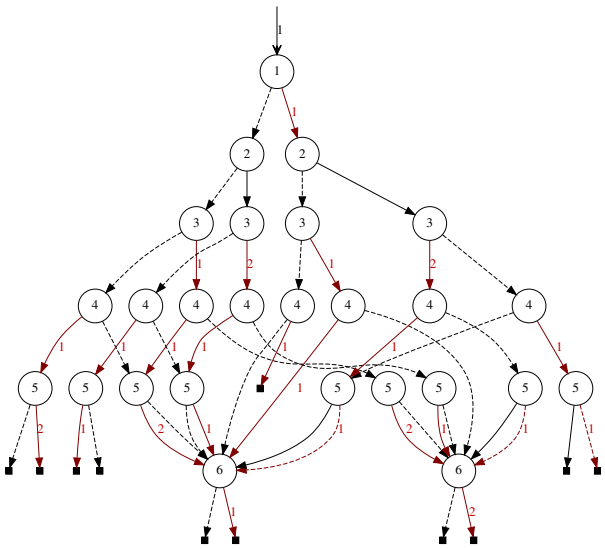

図 2: 合成された WDD

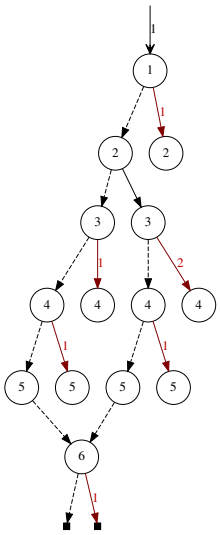

図 3: 合成後に探索した結果

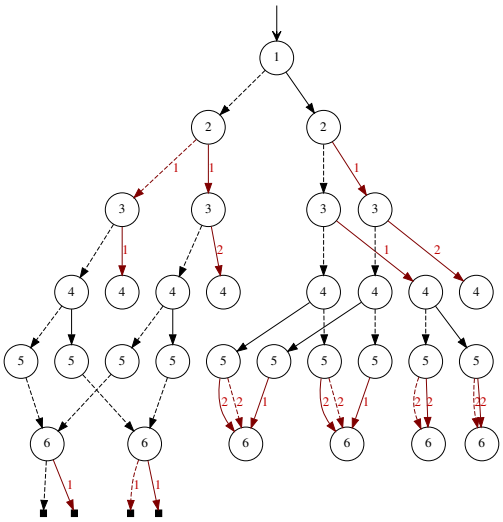

図 4: 合成せず同時探索した結果

に，全ての変数は順序付けされている．終端節点以外の各節点は変数に対応し，その値に対応した 0-枝（破線で表示）と 1-枝（実線で表示）の二つの出力枝を持っている．枝はそれぞれ重みを持っており，図では値 0 の重みの表示を省略している．一般的な BDD とは異なり，終端節点は 1 種類のみである（見やすさを考慮して，図では終端節点を一つにまとめていない）．全ての変数の値を決めると，それぞれの WDD 上で根から終端までの経路が定まる．経路上の重みを全て足し合わせた値が，その WDD が表現する関数の出力値である．

WDD では，関数としての意味を変えることなく，節点の出力側の枝重みを入力側に移すあるいはその逆の操作をすることが可能である．したがって等価な関数に対する WDD の形が一意に定まるためには，追加の規則が必要になる．これにはいくつかの方式があり，初期に EVBDD (edge-valued BDD) という名前で提案されたものは，全ての 0-枝の重みが 0 になるように重みを配分していた [Lai 92, Lai 94]．我々は，組合せ最適化問題を解くために近年用いられている次の重み配分規則 [Hooker 13] を採用する．

- 節点から出る枝の重みは非負とし，その少なくとも一方は 0 とする．

ただし，根を示す特別な枝に限っては負の重みを許す．WDD にこの規則を適用するには，各節点の出力側の 2 枝が持つ重み w_0, w_1 のうち小さい方の値 $\min\{w_0, w_1\}$ を出力側の各重みから減じ入力側の各重みに加算する操作を，下位の（終端に近い）節点から順に適用すれば良い．

このように重み配分された WDD には，どの節点からも終端節点に至る重み 0 の経路が一つ以上存在する．節点から出る枝の重みはどれも非負なので，重み 0 の経路が関数の最小値を与える値割り当てに対応する．したがって最小化問題が単一の WDD で表現されたなら，それは既に解けているに等しい．最小値は根の枝に付けられた重みと同じである．

3. 従来の解法

それでは，図 1 に示された 7 個の関数の合計値を最小化する問題について考えよう．一つの素朴な解法は，WDD 演算によって単一の WDD に合成する方法である．7 個の WDD を加算した結果が図 2 に示されている．終端節点を除いて合計 15 節点であったものが 26 節点に増加した．単一の WDD ができてしまえば，その上位節点から順に重み 0 の分岐を選択していくだけで，手戻りなく最適解を見つけることができる．その結果は図 3 となり，最小値 1 を持つ最適解が 2 通りあることがわかる．このように，合成された WDD では無駄な探索が発生しないが，探索されない部分のグラフ構造まで事前に構築している点が無駄であるように見える．またこの方法は，WDD サイズの爆発により複雑な関数への適用が現実的ではない．

もう一つの解法は，与えられた WDD 群のままそれらを同時に探索することで WDD サイズの爆発を避ける方法である．同じ問題に対する探索の様子を図 4 に示した．探索の初期状態は，与えられた各 WDD の最上位の節点で構成される節点集合である．各探索状態から最上位変数の値を 0 に決めた場合と 1 に決めた場合を考えて節点集合を更新し，二つの次状態を計算する．そのとき元の WDD 上で通過した枝の重みの合計を，次状態に至る枝の重みとする．こうして構築される新しいグラフは負の重みを持たないので，ダイクストラ法を使って効率良く最短経路を求めることができる．すなわち，未処理の状態群の中でそこに至る経路が最短なものを選んで次状態を求める操作を繰り返せば，探索範囲を最小限に抑えることが可能である．図 4 は，対象のグラフの全ての最短路を求める問題としては，全く無駄のない探索の結果である．しかしながら前処理で合成した WDD の図 2 と比較すると，探索対象であるグラフの重みの付き方が異なっている．同時探索では重みが全体的に下位の枝に現れるため枝刈りが遅れ，その先に進んでも最適解に到達しない節点まで広く探索することになってしまう．

4. 提案手法

WDD の合成（加算）はサイズ爆発の危険性を持つ一方で，枝の重みをより上位に持ち上げて枝刈り判定を早める効果を持っている．例えば図 1 の WDD 群では，レベル 4 の節点から出る枝（「レベル 4 の枝」と呼ぶ）に非 0 の重みは存在しない．これに対して f_1 と f_2 を加算すると図 5 の WDD が得られ，レベル 4 の枝に非 0 の重みが現われている．これは，f_1 におけるレベル 6 の 1-枝の重みと f_2 におけるレベル 6 の 0-枝の重みが組み合わされた結果である．一般に，同じレベルに重

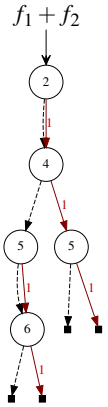

図 5: 単純な加算

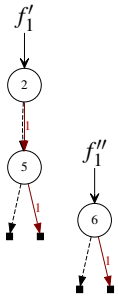

図 6: 加法分解によるレベル 6 の枝重みの分離

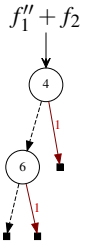

図 7: レベル 6 の枝だけに重みを持つ WDD の加算

み成分を持つ複数の WDD を加算することで重みの持ち上げが起こる．提案手法ではこのことに着目して，特定レベルの重み成分だけを持つコンパクトな WDD を加算することで，サイズ爆発を抑えながら重みの持ち上げをはかる．

具体的には，それぞれの WDD を二つに複製して一方はレベル k の枝重みを 0 に，他方はレベル k 以外の枝重みを 0 にすることで，両者の和が元の WDD と同じ関数を表すように分解（加法分解）する．結果はどちらの WDD も元の WDD 以下のサイズを持ったものになる．図 6 の f'_1 と f''_1 は，$k=6$ として f_1 を加法分解したものである．レベル 6 の枝だけに重みを持った f''_1 と f_2 を加算すると，図 7 のように重みがレベル 4 の枝まで持ち上がる．レベル k の枝重みを持つ WDD を加算して一つに合成するとレベル k からの重み持ち上げを網羅的に計算することができるので，これを下位から上位のレベル k について繰り返せば目的の等価変換が完了する．

この等価変換を図 1 の問題に適用した結果の WDD 群が図 8，それらを同時探索した結果が図 9 である．ここで，探索の範囲が図 4 と一致していることに注目してほしい．一般にこの等価変換を適用すると，図 8 のように，どのレベルについても非 0 の枝重みが高々一つの WDD にしか現れない WDD 群を得ることができる．そのような WDD 群では同時探索におけるどの状態からも重み 0 の分岐が存在することが保証され，単一の WDD に合成した場合と同じ探索範囲で最適解を見つけることができる．

残念ながら，この等価変換は常に可能なわけではない．我々の実験では，途中あるいは最終的な個々の WDD のサイズが肥大化して処理を継続できないケースが多く確認された．しかし，等価変換は最適解発見のコストを削減するための前処理であり，正確に上の通り実行しなくても，関数の和として等価性を保っている限り解の最適性には影響しない．そこで我々は，サイズがあるしきい値 M を超える WDD を処理対象から除外して処理を継続する方法（アルゴリズム 1）を試みた．

アルゴリズム 1 は，パラメータ M をしきい値として集合 $F=\{f_1,f_2,\ldots,f_m\}$ を等価変換する．WDD が示す関数は n 個の変数を含んでおり，WDD で最上位の変数をレベル 1，最下位の変数をレベル n としている．最も外側のループは，下位から順にレベル k について処理を繰り返す．各レベルの処理において，基本的に G はそのレベルに重みを持たない WDD の集合，h はそのレベルに重みを持つ WDD を加算して作られた WDD である．ただし例外として，加算してもサイズ爆発につながらない定数 WDD は G に追加する代わりに h に加算し，加算するとサイズの大きくなる WDD は加算せず G に追加している．アルゴリズムの中で，$\texttt{decompose}(f,k)$ は f をレベル k に枝重みを持たない f' とレベル k 以外に枝重みを持たない f'' に加法分解する関数，$\texttt{is\_const}(f)$ は f が定数を表すことを判定する関数，$\texttt{size}(f)$ は f の非終端節点の数を求める関数である．レベル k に重みを持つ f'' を h に加算しようとするとき（9 行目），結果のサイズが M より小さければそのまま加算するが，そうでない場合は f'' と h の間でサイズの大きい方を加算対象から外して G に保存し，小さい方を h として残している．

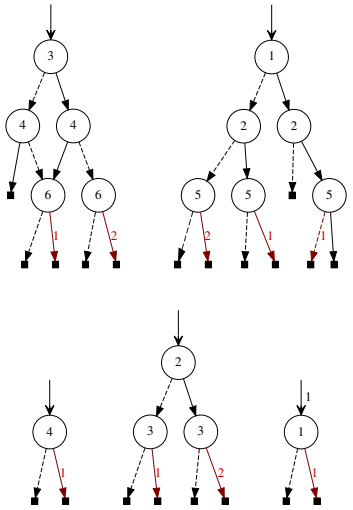

図 8: 等価変換後の WDD 群

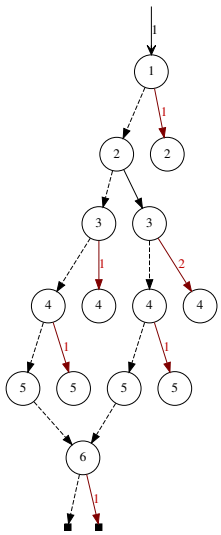

図 9: 等価変換後に同時探索した結果

アルゴリズム 1: WDD 群 $F=\{f_1,f_2,\ldots,f_m\}$ の等価変換

```
1  for k ← n downto 1 do
2      G ← {}, h ← 0
3      foreach f ∈ F do
4          (f', f'') ← decompose(f, k)
5          if is_const(f') then
6              h ← h + f'
7          else
8              G ← G ∪ {f'}
9          if size(h + f'') < M then
10             h ← h + f''
11         else
12             if size(f'') < size(h) then
13                 (h, f'') ← (f'', h)
14             G ← G ∪ {f''}
15     F ← G ∪ {h}
```

5. 実験結果

MaxSAT-2006 ベンチマーク [*2] から単一の WDD が構築できる程度の比較的小規模な例題を選び，各手法の特徴を確認した．表 1 は最適解探索の対象となる WDD の合計サイズ（非終端節点数の合計）を示したものである．n は例題における変

*2 http://www.iiia.csic.es/maxsat06/

表 1: WDD サイズの合計

例題			無変換	提案手法			合成
	n	m		$M=100$	$M=10000$	$M=\infty$	
t3g3-5555	27	162	324	2245	87634	104333	782911
johnson8-2-4	28	420	840	6566	180768	147735	239587
wqueens6_4	36	230	460	1868	60030	60030	4415828
san1000	40	744	1488	12257	632471	1395457	11300925
vcsp21_4_33_91_1	42	1035	4140	2382	3014	3014	250752
29	139	711	1378	9634	540284	708824	95363753

表 2: 探索空間の大きさ

例題			無変換	提案手法			合成
	n	m		$M=100$	$M=10000$	$M=\infty$	
t3g3-5555	27	162	59063	7303	1516	78	78
johnson8-2-4	28	420	16999772	559863	26048	1045	1045
wqueens6_4	36	230	8655	5088	71	71	71
san1000	40	744	*timeout*	70928158	2788389	117	117
vcsp21_4_33_91_1	42	1035	57788219	10221	196	196	196
29	139	711	7431044	142644	381452	549	549

表 3: CPU 時間（秒）

例題			無変換	提案手法			合成
	n	m		$M=100$	$M=10000$	$M=\infty$	
t3g3-5555	27	162	0.1	0.1	0.5	0.9	1.4
johnson8-2-4	28	420	50.5	0.7	1.8	2.3	8.0
wqueens6_4	36	230	0.1	0.1	0.4	0.4	17.1
san1000	40	744	>3600	122.6	14.2	59.6	42.3
vcsp21_4_33_91_1	42	1035	304.5	0.2	0.2	0.2	3.1
29	139	711	8.4	0.5	20.1	59.4	281.3

数の数，m は関数の数を表している．「無変換」は入力された m 個の WDD,「提案手法」はしきい値 M を 100，10000，および無限大に設定したときの等価変換の結果,「合成」は単一の WDD に合成した結果の WDD サイズを示している．提案手法による等価変換では元の WDD 群よりもサイズが大きくなる傾向があり（小さくなっているものもある），増大の度合いは M の値でコントロールできている．また，サイズ制限をしない $M=\infty$ でも単一の WDD に合成したものほどは大きくならなかった．

表 2 は，全ての最適解を列挙するためにダイクストラ法で探索された空間の大きさを示している．これには探索対象である WDD のサイズ（表 1）とは反対の傾向が見られる．また，提案手法で $M=\infty$ のときと単一の WDD に合成したときとでは探索空間の大きさが一致することも確認できる．なお，例題 san1000 の無変換での探索については，1 時間以内に最適解を見つけることができなかった．

前処理と最適解探索を合わせた全体の処理時間が表 3 に示されている．単一の WDD に合成する手法は WDD サイズが大きくなるため前処理に時間がかかり，変換せず同時探索する手法は探索空間が大きくなるため探索に時間がかかる．提案手法はそのトレードオフを調整する仕組みと見ることができ，どの例題に対しても提案手法のどれかが最も高速に問題を解いている．しかしながら，最適な M の値を事前に予測することは難しい．その点において，より洗練された手法の開発が必要であろう．

6. おわりに

組合せ最適化問題の二分決定グラフに基づく解法を，目的関数や制約条件を単一のグラフに統合することが困難な問題に拡大する一つの手法を紹介した．WDD は線形／非線形の目的関数と BDD で表現されるような制約条件を自然に組み合わせることのできるデータ構造である．重みを上位に持ち上げるという単純な目標によって最適解探索のコストを改善する枠組みには，既存研究との関連性と既存研究にない可能性が多くあるように思える．本研究では加法分解された目的関数がコンパクトな WDD 群として与えられる問題に焦点を当てたが，乗法など他の分解方法について考えてみることも興味深い．

参考文献

[Behle 07] Behle, M.: On Threshold BDDs and the Optimal Variable Ordering Problem, *Journal of Combinatorial Optimization*, Vol. 16, No. 2, pp. 107–118 (2007)

[Bryant 86] Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, Vol. 100, No. 8, pp. 677–691 (1986)

[ERAT 14] 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 (2014)

[Hooker 13] Hooker, J. N.: Decision Diagrams and Dynamic Programming, in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 94–110, Springer (2013)

[Inoue 14a] Inoue, T., Iwashita, H., Kawahara, J., and Minato, S.-i.: Graphillion: Software Library for Very Large Sets of Labeled Graphs, *International Journal on Software Tools for Technology Transfer*, Vol. 18, No. 1, pp. 57–66 (2014)

[Inoue 14b] Inoue, T., Takano, K., Watanabe, T., Kawahara, J., Yoshinaka, R., Kishimoto, A., Tsuda, K., Minato, S.-i., and Hayashi, Y.: Distribution loss minimization with guaranteed error bound, *Smart Grid, IEEE Transactions on*, Vol. 5, No. 1, pp. 102–111 (2014)

[Iwashita 13] Iwashita, H. and Minato, S.: Efficient Top-Down ZDD Construction Techniques Using Recursive Specifications, Technical Report TCS-TR-A-13-69, Division of Computer Science, Graduate School of Information Science and Technology, Hokkaido University (2013)

[Kawahara 14] Kawahara, J., Inoue, T., Iwashita, H., and Minato, S.: Frontier-based Search for Enumerating All Constrained Subgraphs with Compressed Representation, Technical Report TCS-TR-A-14-76, Division of Computer Science, Graduate School of Information Science and Technology, Hokkaido University (2014)

[Knuth 11] Knuth, D. E.: *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms, Part 1*, Addison-Wesley Professional, 1st edition (2011)

[Lai 92] Lai, Y.-T. and Sastry, S.: Edge-Valued Binary Decision Diagrams for Multi-level Hierarchical Verification, in *Proceedings of the 29th ACM/IEEE Design Automation Conference*, DAC '92, pp. 608–613, Los Alamitos, CA, USA (1992), IEEE Computer Society Press

[Lai 94] Lai, Y.-T., Pedram, M., and Vrudhula, S. B. K.: EVBDD-Based Algorithms for Integer Linear Programming, Spectral Transformation, and Function Decomposition, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 8, pp. 959–975 (1994)

[Minato 93] Minato, S.: Zero-suppressed BDDs for Set Manipulation in Combinatorial Problems, in *Proceedings of the 30th Design Automation Conference*, DAC '93, pp. 272–277 (1993)

[Nishino 15] Nishino, M., Yasuda, N., Minato, S., and Nagata, M.: BDD-Constrained Search: A Unified Approach to Constrained Shortest Path Problems, in *AAAI Conference on Artificial Intelligence* (2015)

[Toda 15] Toda, T. and Soh, T.: Implementing Efficient All Solutions SAT Solvers, *CoRR*, Vol. abs/1510.00523, (2015)

[柳谷 93] 柳谷 雅之：組合せ最適化問題の BDD による解法, 情報処理, Vol. 34, No. 5, pp. 617–623 (1993)