1F3-3

ZDDを用いた多次元ナップサック問題の厳密解法

An Exact Algorithm for Multi-Dimensional Knapsack Problem using ZDD

安田宜仁 *1　西野正彬 *2　湊真一 *1
Norihito YASUDA　Masaaki NISHINO　Shin-ichi MINATO

*1北海道大学大学院情報科学研究科
Graduate School of Information Science and Technology, Hokkaido Univercity

*2日本電信電話 (株) NTT コミュニケーション科学基礎研究所
NTT Communication Science Laboratories., NTT Corp.

It is well known that decision diagrams (xDDs) can be used for solving 0-1 intger linear programming problems. However, due to the explosion of the its size, xDDs have been considered unattractive for solving this kind of problems. However, it is recently shown that xDD-based approaches are powerful and scalable in cases where constrains are derived from graph sub-structures. Following this success, we think that it is imortant to explore the potential power of xDD based approaches for ILP again. To this end, this paper shows a ZDD-based optimization algorithm that treats vanilla setting of multi-dimensional 0-1 knapsack problem, a variant of knapsack problem. Naive approaches using xDDs first construct a diagram that contains all the possibile assignments that satisfy the given constraints. However, since our purpose is obtaining an optimal solution, we do not have to maintain all the possible assignments. Therefore, we propose a novel node trimming method that assures optimality of the obtained solution while guaranteeing to never increase the number of nodes in the diagram.

1. はじめに

よく知られている通り，二分決定グラフ (BDD)[Bryant 86] やゼロサプレス型二分決定グラフ (ZDD)[Minato 93] は 0-1 整数線形計画問題を解くための道具として用いることができる．具体的には，制約条件に対応する決定グラフを構築し，目的関数の係数を 1-枝の重みとする重みつき決定グラフを考え，その上での最短路あるいは最長路を求めることで 0-1 整数計画問題を解くことができる．この方法は決定グラフのサイズに対して線形の計算時間で最適解を求めることができるものの，決定グラフの大きさが場合によっては指数的に増大することを考えると他の解法に比べあまり魅力がなかった．

しかし，近年，変数間にグラフ構造由来の制約条件を考える問題において，決定グラフを用いることが効果的であることが示されている．たとえば，文献 [Nishino 15] では，アイテムの間に木構造を成す状況において，木の形を保ったままアイテムを選択する (1 次元) ナップサック問題に対して，ZDD を用いた手法が有効であることが示されている．あるいは，文献 [Inoue 14] では電力配電網の構成が全域森でなければならないという制約下での，スイッチ割り当て最適化問題を ZDD を用いて問いている．このようなグラフ構造制約は従来のアプローチでは取扱いが難しい．汎用 ILP ソルバを用いようにも，ソルバに与える制約式の数自体が指数的に増えてしまう場合がある．このような背景を考慮した場合，0-1 整数計画問題に対して決定グラフを用いるアプローチは改めて探究すべき対象だと思われる．そこで本稿ではまず，グラフ構造制約のように決定グラフにとって得意な条件設定ではなく，基本的な設定の 0-1 整数計画問題について，改めて決定グラフを利用した解法を探る．

素朴に決定グラフを用いるアプローチを振り替えると，求解の中間生成物として，制約を満たすようなすべての変数割り当てパターンを保持するような決定グラフを一旦作ることになる．制約式が複数の式から構成される場合，それ ぞれの制約式の AND 条件となってしまい制約条件を表現する決定グラフのノード数が指数的に増大しがちである．制約条件を表現する決定グラフとは，言い替えれば，線形不等式で表現された制約式をすべて満たすような変数の割り当て方をすべて保持したような決定グラフである．0-1 整数計画問題においては「ただひとつの最適解」が欲しいわけであるから，問題設定に比べてその中間生成物としての二部決定グラフが過剰に情報を保持している感は否めない．

そこで本稿では決定グラフのサイズを抑えながら求解する方法を考える．中間生成物としての決定グラフのサイズを抑えるという意味では，厳密に決定グラフを作成する代わりに，制約条件を逸脱する割り当てを許容した決定グラフや，逆に，制約条件を満たす割り当ての一部のみを含む決定グラフを用いる方法が提案されている [Kell 13, Bergman 16]．本稿では多くの問題においては高速な近似解法が存在することに着目し，近似解をヒントとして決定グラフのサイズを抑えることを試みる．この例題として本稿では多次元ナップサック問題を取り扱う．多次元ナップサック問題を取り扱う理由は 2 つある．まず，制約条件複数存在するため，すべての条件を満たすような決定グラフのノード数が爆発しがちで素朴な決定グラフのアプローチでは困難であることが挙げられる．2 つめの理由として，多次元ナップサック問題は各種メタヒューリスティックの実験場のような側面があり，優れた近似解法が多数存在することが挙げられる．

提案法では近似解を用いて枝刈りを行う．この際，決定グラフが表現している解候補集合を絞り込むのではなく，決定グラフを通常の有効非巡回グラフ (DAG) として見た時に，最適解のパスが通ることはないノードを削除するのが特徴である．

連絡先: yasuda@ist.hokudai.ac.jp

2. 準備

2.1 多次元ナップサック問題

m 次元 n アイテムの 0-1 多次元ナップサック問題は，以下の 0-1 整数計画問題として表現できる:

$$\begin{aligned} \text{最大化} \quad & \sum_{j=1}^{n} p_j x_j, \\ s.t. \quad & \sum_{j=1}^{n} w_{i,j} x_j \le c_i, \quad i = 1, ..., m \\ & x_j \in \{0, 1\}, \quad j = 1, ..., n \end{aligned}$$

通例に従い，p_i をアイテムの価値，$w_{i,j}$ をアイテムの重さ，c_i を重さ制限と呼ぶ．

2.2 BDD・ZDD

BDD は DAG を用いた論理関数の表現である．論理関数の値を全ての全ての変数について場合分けした結果を表現した二分決定木を縮約することによって得られる．

ZDD は組み合わせの集合を表現することに特化した BDD の亜種である．BDD 同様に ZDD も DAG で表現され，ひとつの根ノード，2 つの終端ノード (0-終端，1-終端) と内部ノードから構成される．内部ノードはすべて 0-枝と 1-枝と呼ばれるふたつのエッジを持つ．内部ノードは組合せを構成する変数と対応付いており，事前に固定された変数の順序に沿うように DAG 中での順序が決まっている．ZDD が保持している組合せは，根ノードから 1-終端までのパスによって表現される．パス中で，1-枝を通ったノードに対応する変数が組合せの中のアイテムに相当する．たとえば，図 1(左) は {{x1},{x1, x4}, {x2}, {x2, x4}, {x3}, {x3,x4}, {x4}, {}} という組合せの集合を保持した ZDD である．BDD・ZDD を縮約するたの規則として 1) 冗長なノードの削除，2) 等価なノードの共有が存在する．これらの規則をできる限り適用した形は一意に定まり既約な決定グラフと呼ばれる．決定グラフの魅力として，論理関数や組合せの集合を圧縮した表現のまま各種演算が行える点が挙げられる．本稿では特に，ZDD の交差演算 (ふたつの ZDD を入力として，双方に含まれるような組合せのみからなる ZDD を返す演算) を利用する．

3. ZDD による解法

3.1 線形不等式制約に対する ZDD の構築

Knuth による単純パスの列挙アルゴリズム [Knuth 11] 以降，グラフの部分構造列挙を中心にさまざまな対象に対して，ノード共有済みの ZDD を根ノードからトップダウンに直接構築できるようになってきている．本稿で用いる線形不等式に対する ZDD もトップダウン構築が可能である．

トップダウン構築のためには，1) どういう状況であれば同じレベルのふたつのノードを等価とみなすかという条件と，2) どういう状況であればそのノードから下位の構築を行わずに終端するかという条件を決定する必要がある．まず，等価とみなす条件について述べる．不等式制約では，変数の 1-枝にアイテムの重み和を対応づけた場合の，ノードまでの重みの和が同一であればその後に選択可能なアイテム集合は同じなので等価とみなすことができる．次に，終端の条件については，重みの和が閾値 (制約の右辺) を越えてしまった場合，それ以下のアイテムを選択することはできないので，当該ノードからの 1-枝を 0-終端へ接続する．

図 1(右) に不等式制約 $x_1 + 2x_2 + 3x_3 + 6x_4 + 8x_5 \le 12$ を満たすようなすべての組合せを含んだ ZDD を示す．

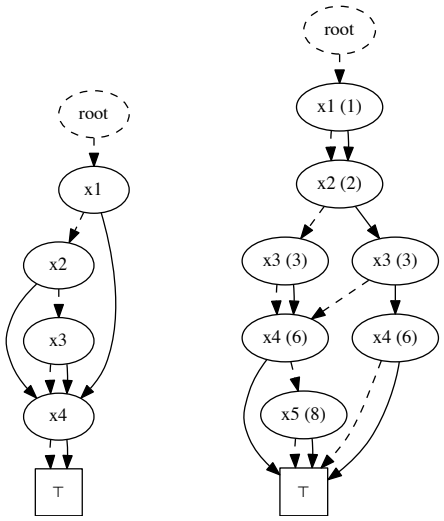

図 1: ZDD の例 (左) と不等式制約を示す ZDD(右)

Algorithm 1 ZDD を用いた多次元ナップサック問題求解

```
Q ⇐ 空のキュー
for c in constraints do
  Z_c ⇐ GenZDD(c)
  Enqueue(Q, Z_c)
end for
while Q の要素数 > 1 do
  Z_0 ⇐ Dequeue(Q)
  Z_1 ⇐ Dequeue(Q)
  Z_t ⇐ Z_0 ∩ Z_1
  Enqueue(Q, Z_t)
end while
Z_a ⇐ Dequeue(Q)
アイテムの価値を Z_a の各 1-枝に関連付け
LongestPath(Z_a)
```

3.2 基本となる解法

提案法について述べる前に，ZDD を用いて多次元ナップサック問題を解く場合の基本となる解法について述べる．アルゴリズムを Algorithm 1 に示す．大まかには，すべての制約条件を満たすような変数の組合せを保持した ZDD を構築し，その ZDD の上での最長路を求めている．関数 GenZDD() は不等式制約を引数として，3.1 節で述べたトップダウン構築によって ZDD を作成する関数である．関数 LongestPath() は 1-枝の重みをアイテムの価値とした ZDD を引数としてその最長路を返す関数である．∩ は交差演算を示す．

交差演算の結果のノード数は元のノード数の積で大きくなることがあるため，ひとつの ZDD に順次交差演算を行っていくのではなく，2 つの交差の結果をまず作り，次に 4 つの交差を作るといった順で構築している．なお，キューの代わりにスタックを用いれば順次作成することになる．

3.3 提案するノード削除手法

本稿で提案するノード削除は，ZDD のセマンティクスを考慮せずに DAG として見た時に，解が決して通ることのないようなノードを削除する方法である．アルゴリズムを Algorithm 2 に示す．Addr() はノードのアドレス (0 からノード数の間の数値) を返す関数である．Hi()，Lo() はそれぞれ 1-枝，0-枝の

指し先のノードを返す関数である．ZDD 中の 0-枝の重さを 0，1-枝の重さをその変数に対応するアイテムの価値としたような重みつきのグラフを考え，各ノードに対応する値を以下の手順で求める．まず，根ノードから各ノードまでの最長路をたどった場合の距離 (価値の和) を LT に記録する．次に，各ノードから 1-終端までの最長路をたどった場合の距離を LB に記録する．LT の値と LB の値の和が近似解未満であるようなノードは，そのノードを通るような任意のパスは最適解にはならないことを意味する．そのようなノードは最適化問題を解くという意味では ZDD から削除しても解に影響を与えないので削除対象として DE に印をつける．次に，再度すべてのノードを走査し，枝の指し先が削除対象ノードであれば差し先を 0-終端に張り替える．最後に既存の簡約化アルゴリズム (文献 [Knuth 11] の Algorithm R) を実行し既約な ZDD を得る．なお，ここでは削除対象のノードや，削除対象ノードから参照されていたノードについて明示的な削除処理は行わない．通常の ZDD の演算同様，一時的に生じたごみノードとしてガベージコレクションの際に回収される．

上記のノード削除は ZDD が保持している組合せ集合を操作するのではなく，DAG としてのグラフを直接操作するため，削除後にノード数が増えることがないという特徴がある．もし組合せ集合を操作し，近似解に満たないような組合せを削除しようとした場合，削除後のノード数を測することは難しい．

ノード削除を行いながら求解する手順をアルゴリズムを Algorithm 3 に示す．関数 Elim() は上で定義したノード削除である．この手順では最初にヒューリスティック手法等で近似解を求める．各制約条件に対する ZDD を構築した後，および交差演算を行った後にノード削除を実行する．

Algorithm 2 ノード削除 Elim(Z, r)

```
VR ⇐ 各変数 (各レベル) の価値を含んだ配列
LT, LB, DE ⇐ ノード数個要素の配列を 0 で初期化
for node in (根ノードとレベル降順の内部ノード) do
  lv ⇐ ノードのレベル
  if LT[Addr(node)] > LT[Addr(Hi(node))] + VR[lv]
  then
    LT[Addr(Hi(node))] ⇐ LT[Addr(node)] + VR[lv]
  end if
  if LT[Addr(node)] > LT[Addr(Lo(node))] then
    LT[Addr(Lo(node))] ⇐ LT[Addr(node)]
  end if
end for
for node in (レベル昇順の内部ノード) do
  lv ⇐ ノードのレベル
  if LB[Addr(Hi(node))] + VR[lv] > LB[Addr(node)]
  then
    LB[Addr(node) ⇐ Addr(Hi(node))] + VR[lv]
  end if
  if LB[Addr(Lo(node))] > LB[Addr(node)] then
    LB[Addr(node) ⇐ Addr(Lo(node))]
  end if
end for
for node in (内部ノード) do
  if LB[Addr(node)] + LT[Addr(node)] < r then
    DE[Addr(node)] ⇐ 1
  end if
end for
for node in (内部ノード) do
  if DE[Addr(Hi(node))] = 1 then
    Hi(node) ⇐ 0-終端
  end if
  if DE[Addr(Lo(node)] = 1 then
    Lo(node) ⇐ 0-終端
  end if
end for
簡約化アルゴリズムを実行
```

4. 評価

4.1 実験条件

提案するノード削除を行わない場合，および整数計画法の商用ソルバとの比較を行った．

テストセットには，OR-Library[*1] から入手可能なテストセットのうち，“mknap1.txt” と “mknap2.txt” の計 54 問を用いた．これらは定義通りの多次元ナップサック問題であり，その他の制約は含まれていない．変数数 n の範囲は 10 から 90，次元数 m の範囲は 2 から 50 であり，現在の ILP ソルバにとっては厳密解を得ることは容易な問題セットである．

ZDD の作成には C++ライブラリ TdZdd[tdz] を用いた．TdZdd を用いることにより，3.1 節で述べた 2 つの条件を指定するだけで，容易に ZDD を構築することができる．その他の部分を含めて ZDD を用いた求解はすべて C++で実装した．Algorithm 2 で用いる近似解を得る方法として，アイテムをランダムに選択し続ける方法を用いた．この操作を 65536 回行った結果得られた解のうちもっとも良い解を近似解とした (多次元ナップサック問題には優れた近似解法が多数存在するので，この方法はお勧めするものではない)．商用整数計画ソルバには，Gurobi 6.5.1 を用いた．実験に使った計算機は CPU Core i5 2.9 GHz，メモリ 16 GB の MacBook Pro である．CPU 時間の上限を 30 秒とした．

4.2 実験結果・考察

実験結果を表 1 に示す．表より，提案法はノード削除を行わない場合に比べて所定の時間内で解けた問題数が 25 問から 31 問へ 6 問増加していることが分かる．また，解けた問題についても概ね数倍から数十倍高速化されている．提案法は DAG の走査やノード削除の処理を行うが，それを差し引いてもなお高速化が可能であったと言える．

とはいえ，30 秒の CPU 時間制限で解けた問題は 54 問中 31 問に留まり，全ての問題を高速に解いている ILP ソルバと比べると依然として格段に遅いと言わざるを得ない．特に，変数の数が 50 を越えるとその差はさらに広がり，ZDD を用いた手法は所定の時間で解を得ることが難しくなっている．

しかし，これはある程度は事前に予測された結果であり，前述した通り，グラフの構造由来の制約のような書き下すことが本質的に困難な種類の制約の取扱いも視野に入れた場合には，ILP ソルバに比べて xDD を用いた手法が有効であることが示されている．これらの従来法では，提案法のようなノード削除は行っておらず，表中の「ノード削除なし」に相当する処理を行っていることになる．提案法のノード削除と組合せることで，グラフの構造制約を伴う整数計画問題をより効率的に解ける可能性がある．

*1 http://people.brunel.ac.uk/ mastjjb/jeb/info.html

Algorithm 3 ノード削除を行う多次元ナップサック問題求解

$Q \Leftarrow$ 空のキュー
$r \Leftarrow$ 近似解 (厳密解以下であること)
for c in constraints **do**
  $Z_c \Leftarrow \mathrm{GenZDD}(c)$
  $\mathrm{Elim}(Z_c, r)$
  $\mathrm{Enqueue}(Q, Z_c)$
end for
while Q の要素数 > 1 **do**
  $Z_0 \Leftarrow \mathrm{Dequeue}(Q)$
  $Z_1 \Leftarrow \mathrm{Dequeue}(Q)$
  $Z_t \Leftarrow Z_0 \cap Z_1$
  $\mathrm{Elim}(Z_t, r)$
  $\mathrm{Enqueue}(Q, Z_t)$
end while
$Z_a \Leftarrow \mathrm{Dequeue}(Q)$
アイテムの価値を Z_a の各 1-枝に関連付け
$\mathrm{LongestPath}(Z_a)$

5. まとめと今後の課題

0-1 整数線形計画問題，特に多次元ナップサック問題について，ZDD を用いて解く際の性能を評価し，新しいノード削除方法を提案した．提案した方法は ZDD のセマンティクスを考慮せずに，根ノードからノードへの最長路と，ノードから 1-終端までの最長路の和が近似解を越えられるかどうかを用いてノード削除を行うのが特徴である．実験では，ノード削除を行わない場合に比べて最大数十倍高速化できることを示した．今回単純な問題設定において評価を行ったが，近年 ZDD を用いた整数計画問題解法はグラフ構造制約との組合せにおいて成功を収めており，グラフ構造制約を持つ条件下での提案法の有効性を今後検証したい．

謝辞

本研究の一部は科研費基盤 (S)15H05711 の助成による．

参考文献

表 1: 実験結果 (単位は秒,「-」は 30 秒以上の計算時間を示す)

問題 (n-m-最適解)	提案法	ノード削除なし	ILP ソルバ
10-10-8706.1	0.027	0.035	0.006
15-10-4015	0.045	0.062	0.011
20-10-6120	0.056	0.106	0.005
28-10-12400	0.128	0.459	0.015
39-5-10618	0.282	1.932	0.030
50-5-16537	0.860	16.124	0.040
6-10-3800	0.013	0.018	0.004
105-2-1095445	7.565	-	0.008
105-2-624319	4.537	4.670	0.017
20-10-2139	0.177	0.303	0.035
27-4-3090	0.062	0.148	0.015
28-2-119337	0.049	0.106	0.005
28-2-130623	0.043	0.147	0.006
28-2-130883	0.044	0.140	0.006
28-2-141278	0.047	0.177	0.008
28-2-95677	0.046	0.080	0.009
28-2-98796	0.040	0.091	0.004
28-4-3418	0.068	0.179	0.016
29-2-95168	0.048	0.086	0.009
30-5-4115	0.361	1.638	0.007
30-5-4536	0.173	2.214	0.012
30-5-4554	0.118	2.973	0.014
30-5-4561	0.069	0.891	0.004
30-50-4514	0.091	0.914	0.004
34-4-3186	0.124	0.265	0.018
35-4-3186	0.140	0.277	0.026
37-30-1035	-	-	0.094
40-30-776	-	-	0.088
40-5-5246	18.870	-	0.004
40-5-5557	2.138	-	0.017
40-5-5567	1.582	-	0.019
40-5-5605	2.084	-	0.010
50-5-5643	-	-	0.012
50-5-6159	-	-	0.015
50-5-6339	22.710	-	0.020
他 $n \geq 60$ かつ $m \geq 5$ の問題 19 問	すべて-	すべて-	平均 0.026

[Bergman 16] Bergman, D., Cire, A. A., Hoeve, van W., and Hooker, J. N.: Discrete optimization with decision diagrams, *INFORMS Journal on Computing*, Vol. 28, No. 1, pp. 47–66 (2016)

[Bryant 86] Bryant, R. E.: Graph-based algorithms for boolean function manipulation, *Computers, IEEE Transactions on*, Vol. 100, No. 8, pp. 677–691 (1986)

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

[Kell 13] Kell, B. and Hoeve, van W.: An MDD approach to multidimensional bin packing, in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 128–143, Springer (2013)

[Knuth 11] Knuth, D. E.: *The Art of Computer Programming, Volume 4A Combinatorial Algorithms Part 1*, Addison-Wesley Upper Saddle River, New Jersey (2011)

[Minato 93] Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems, in *Design Automation, 1993. 30th Conference on*, pp. 272–277IEEE (1993)

[Nishino 15] Nishino, M., Yasuda, N., Hirao, T., Minato, S., and Nagata, M.: A dynamic programming algorithm for tree trimming-based text summarization, in *Proceedings of NAACL HLT*, pp. 462–471 (2015)

[tdz] TdZdd – A top-down/breadth-first decision diagram manipulation framework, https://github.com/kunisura/TdZdd