5 ZDD を用いた解法

湊　真一 | 北海道大学大学院情報科学研究科

ZDD（ゼロサプレス型二分決定グラフ）は，組合せを要素とする離散的な集合データを計算機上で効率良く扱うためのデータ構造である．ZDD は元々は 1990 年代に LSI 設計自動化の分野で考案され発展した技法であるが，2000 年代以降は，データベース解析，制約充足，組合せ最適化，統計データ処理など，さまざまな用途に広く応用されている．最近，DA シンポジウムにおいてナンバーリンク問題を題材としたアルゴリズムデザインコンテストが開催されることになり，筆者らのグループは ZDD を用いたソルバーを作成して，初回および 2 回目のコンテストに出場した．ZDD は膨大な個数の組合せの集合をコンパクトに索引化できるという特長を持つ．多くのナンバーリンクの問題では，解は唯一またはごく少数個しか存在しないため，必ずしも ZDD の利点を最大限には発揮できていないと思われるが，それでもある程度良い結果が得られており，興味深い部分もある．本稿では，アルゴリズムデザインコンテストにおいて ZDD の技法をどのように適用したかを紹介し，本手法の特長および課題となった点などを紹介する．

組合せ集合の ZDD による表現

組合せ集合とは，「n 個のアイテムから任意個を選ぶ組合せ」を要素とする集合のことを言う．たとえば，a, b, c, d, e の 5 個のアイテムに関する組合せ集合の一例として $\{abc, cde, bd, acde, e\}$ などが作れる．組合せ集合は，組合せ問題の解集合を表現する基本的・汎用的なモデルであり，実問題においては，スイッチの ON/OFF の組合せ，顧客データベース，Web のリンクの表現，システム故障要因の表現等，さまざまな局面で現れる．

ZDD (Zero-suppressed Binary Decision Diagram：ゼロサプレス型二分決定グラフ）は，図 -1(a) に示すような非巡回有向グラフによる組合せ集合の表現方法で，筆者（湊）が 1993 年に考案・命名したデータ構造である．これは，図 -1(b) の**場合分け二分木 (binary decision tree)** を圧縮することにより得られる．この場合分け二分木では，各分岐節点の 1- 枝と 0- 枝は，その節点にラベル付けされたアイテムを選ぶかどうかの場合分けを表し，葉の値 (1/0) はその葉に対応する組合せが集合に属するかどうかを示している．たとえば図 -1(b) において，二分木の根から a=1, b=1, c=0 を選んで進んでいって到達できる葉は組合せ ab に，a=1, b=0, c=1 を選んで到達できる葉は組合せ ac に，a=0, b=0, c=1 を選んで到達できる葉は組合せ c に対応している．この 3 通りの葉が 1 の値を持っているので，それらが集合に属する組合せであることが分かる．場合分け二分木から ZDD を得るための圧縮を行う際には，まず場合分けするアイテムの順序を固定し，以下の 2 つの圧縮規則を可能な限り適用する．

(a)（**冗長節点の削除**）1- 枝が 0 の値を持つ葉を指している場合に，この節点を取り除き，0- 枝の行き先に直結させる（図 -2(a)）．

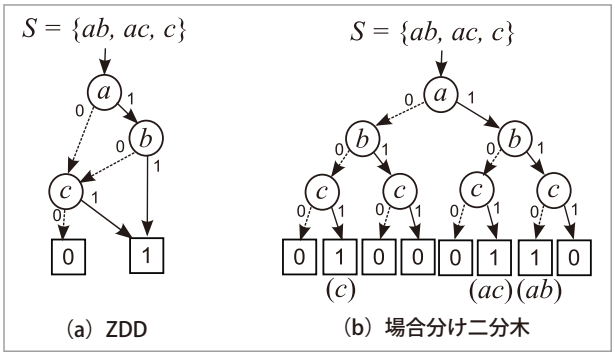

図 -1　ZDD と場合分け二分木

(b)(**等価節点の共有**) 等価な節点 (アイテム名が同じで, 0- 枝同士, 1- 枝同士の行き先が同じ) を共有する. (図 -2(b))

これにより「既約」な形が得られ, 組合せ集合をコンパクトかつ一意に表せることが知られている. ZDD によるデータ圧縮率は, 組合せ集合の性質にも依存するが, 例題によっては数十倍～数百倍以上もの圧縮率が得られる場合がある.

ZDD はデータを圧縮できるだけでなく, 2 つの組合せ集合の間の集合演算を高速に実行できるという重要な特長がある. 図 -3 に概念図を示す. 2 つの既約な ZDD F, G を入力とし, F と G の間の集合演算, たとえば, 交わり (intersection), 結び (union), 差集合 (set difference) などを行い, その結果を表す既約な ZDD H を直接生成するアルゴリズムが知られている. この演算に必要な計算時間は, 多くの場合, 演算結果の ZDD の節点数にほぼ比例するので, ZDD の圧縮率が高ければ高速に集合演算を実行できる. このような集合演算を多段階に実行することにより, 任意の組合せ集合を表す ZDD を構築することができる. また得られた ZDD に対して制約条件を与えて解を絞り込むことができる.

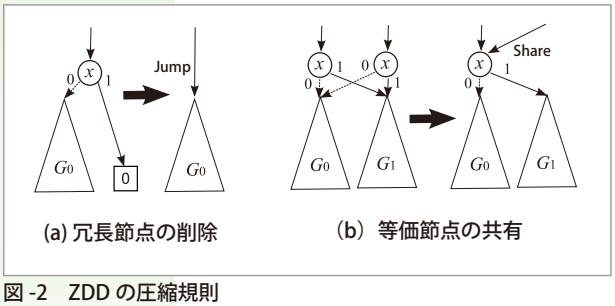

図 -2 ZDD の圧縮規則

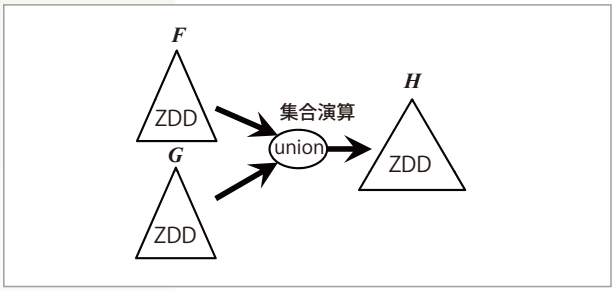

図 -3 ZDD 同士の集合演算

部分グラフ列挙問題と ZDD 表現

ZDD は, グラフの部分集合を列挙索引化するためのデータ構造としても有用である. 頂点集合 $V=\{v_1, v_2, ..., v_n\}$, 辺集合 $E=\{e_1, e_2, ..., e_m\}$ からなるグラフ $G=(V, E)$ を考えると, グラフ列挙の問題とは, E のべき集合 2^E (または V のべき集合 2^V) の中から, ある条件を満たすような部分集合を求めることであり, これは解集合を辺 (または頂点) の組合せの集合と考えれば, そのまま ZDD で表現することができる. たとえば, 図 -4 ではグラフの 2 節点 s, t を結ぶパスの集合を表す ZDD を示している. この例では s, t 間のパスは 4 通り存在するが, それぞれのパスを辺の組合せと考えれば $\{ad, ace, be, bcd\}$ という組合せ集合として解を列挙できる. これを表す ZDD では, 最上位の根の節点から 1 の終端節点に至るパスが 4 通り存在する. ここで, ZDD の 0- 枝はグラフ G の当該辺を使わないことを意味し, 1- 枝は当該辺を使うことを意味すると解釈すれば, ZDD の各パスがこの問題の解に対応していることが確かめられる. この例ではパス集合を表現しているが, パスに限らず, 辺集合で表現できる任意の部分グラフ (サイクル, 木, 森, マッチングなど) を ZDD で列挙できる. ナンバーリンクの解集合もこの枠組みで表現可能である.

フロンティア法による ZDD 生成

Knuth の有名な教科書 The Art of Computter Programming (Vol.4 Fascicle 1) [1] の p.121 (Vol.4A

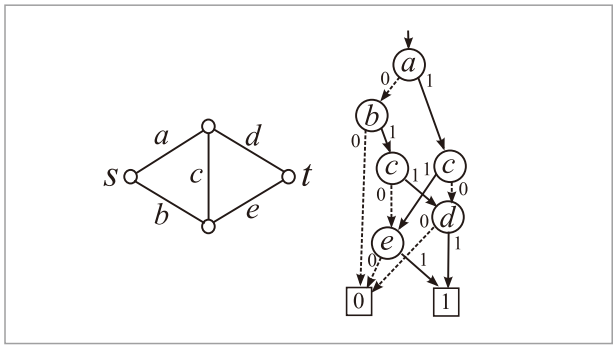

図 -4 s, t 間のパスを列挙する ZDD

では p.254) において，与えられたグラフの 2 点 s, t を結ぶ単純パス（遠回りを許すが同じ頂点は 2 度通らないパス）を全列挙するアルゴリズム Simpath が記載されている．これは，s-t 間の単純パスとなるような辺の組合せを全列挙した ZDD を構築するアルゴリズムである．このアルゴリズムは驚くほど高速であり，たとえば 14 × 14 の格子グラフ（辺の総数 420）の対角 2 頂点を結ぶ単純パスを全列挙する ZDD を生成させると，パスの総数は実に 227,449,714,676,812,739,631,826,459,327,989,863,387,613,323,440（約 2.27×10^{47}）通りもあるが，ZDD の節点数はたった 144,759,636 個で済んでおり，その ZDD 生成と解の数え上げに要する時間はわずか数分である．

図 -5 により Simpath アルゴリズムの大まかな処理手順を示す．まず与えられたグラフ G の各辺に適当な処理順序 (a, b, c, d, e) をつける．これより，根節点から順に a, b, ...の変数順の二分決定木を，各辺を使うか使わないかを場合分けしながら，上位から下位に向かって幅優先順でトップダウンにグラフ断片を生成し，最終的に s-t パスにできるかどうかを分類していく．この幅優先順の処理の各段階で，グラフ G における処理済みの辺と未処理の辺の両方に接続している頂点の集合を Knuth はフロンティア (frontier) と呼び，このフロンティアをグラフ G 上で徐々に移動させながら，フロンティア上の頂点の連結情報を途中状態として記憶して，動的計画法により等価な状態をまとめて共有することにより，圧縮された ZDD を高速に構築している．より詳細については解説記事 [2] を参照されたい）．

Knuth はさらに，Simpath アルゴリズムの一部を変更するだけで，s-t 間の単純パスだけでなく，ハミルトンパス，有向パス，さらに各種のサイクルを列挙する ZDD もほぼ同様に構築できることを述べている．さらに途中状態の記憶の仕組みを改変することで，連結部分グラフの列挙，大域木／林の列挙，カットセットの列挙，グラフの k 分割問題等，多くの問題に適用できる．筆者らはこのような幅優先トップダウン型の動的計画法による ZDD の構築法を総称して「フロンティア法」と呼び，さまざまな実問題への応用を試みている．フロンティア法の処理速度やメモリ量は，処理途中に出現するフロンティアのサイズに大きく依存する．たとえば $n \times n$ の格子グラフの場合，計算途中のフロンティアの幅がなるべく増大しないように辺を適切に順序付けすることにより，非常に圧縮率の高い ZDD を構築することができる．

ナンバーリンクへの適用とコンテストの結果

ナンバーリンクは，各マス目を頂点に，隣接関係を辺に置き換えれば，格子グラフ上で同じ番号ラベルを持つ頂点間の単純パスを探索する問題となる．したがって，複数ペアの始終点を扱えるように一部を改造したフロンティア法を適用すれば解集合の ZDD を構築できる．実はナンバーリンクの問題では，解は 1 通り（または非常に少ない個数）しか存在しないことが多く，必ずしも ZDD の利点を最大限に発揮させる応用先とは言えない．しかし ZDD では，膨大な個数の組合せを辞書順に整理して圧縮した索引構造を作り，高速に

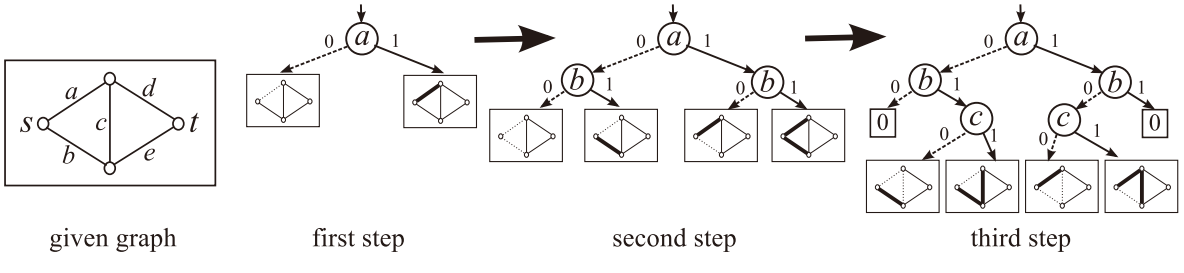

図 -5　Simpath アルゴリズムの処理手順

集合演算を実行できるという特長がある．以下，2 度のコンテストにおいて，ZDD の技法をナンバーリンク問題に適用した様子とその結果を報告する．

第 1 回 (2014 年）のコンテストに参加した際は，事前準備の段階で，単純にフロンティア法を適用しただけでは，例題によっては計算途中の ZDD が大きくなり過ぎてメモリあふれを起こしたり，時間がかかり過ぎる場合があることが分かっていた．そこで以下の 2 つの工夫を盛り込んだ．

- ZDD は変数順序によってサイズが大きく変化する場合があるため，ナンバーリンクの盤面を回転・反転することによって変数順序が変わり，処理効率が変化する場合がある．変数順序の良し悪しは ZDD を生成してみないと判定できないが，始点終点の配置に関するヒューリスティックな方法で比較的良い変数順序を得られるようにした．
- すべての空間を探索すると処理時間がかかり過ぎるので，ナンバーリンクとしてはあり得ない条件を除外する．たとえば「線がコの字型に曲がる」は直線で結ぶ方が明らかに良いので除外するなど．

これらの工夫などにより，我々のチームは準優勝の成績を得ることができた．これは，すべての解を列挙索引化する ZDD を用いた手法が，1 つの解を探索する手法とも競争できるレベルの高い処理性能を持つことが示された形となっている．しかし正直なところ，初回のコンテストではまだ参加各チームの完成度がそれほど高くはなく，途中でプログラムが止まってしまったチームなどもあったと記憶している．実際の現場では，限られた時間内でのパラメータの調整など細かいテクニックも重要である．我々のチームは当時 NTT 研究所から ERATO プロジェクトに出向していた安田宜仁氏を中心として，プログラミングや PC 操作の経験が豊富なメンバがおり，バグ修正やヒューリスティック手法の調整を的確に行えたことも好成績の要因であったと思う．

2 回目 (2015 年) のコンテストでは，ERATO プロジェクトの終了年度にあたっており，学生を中心とするメンバで参加した．前年度と同じことをしても面白くないので，当時，修士課程の鈴木浩史君を中心にして前年度のアルゴリズムに別の工夫を加えたプログラムを開発して挑戦することとした．基本的なアイディアは，トップダウンに幅優先型に ZDD 節点を生成していく途中で，節点数が大きくなり過ぎてメモリあふれを起こして途中終了してしまうのを防ぐため，図 -6 のように ZDD の幅（同一レベルの ZDD 節点数）がある制限値 w を超えた場合に，それ以上は正しい ZDD 節点を生成せず，強制的に 0- 終端に落としてしまうという方法である．これによって与えられたメモリの範囲内で途中終了することなく探索を続けられるようになるが，探索をあきらめた部分空間に解があった場合にはそれを発見できなくなるリスクがある．実際のナンバーリンクの例題に対して幅制限を加えて実行してみると，大規模な例題では探索空間をほとんどカバーできず解を発見できなくなり，実用的な効果は期待できないことが分かった．

そこで次に，図 -7 のように，トップダウンで ZDD 節点を展開する際に k 段（k は適当な定数）下方まで先読みを行い，その先に解が存在する見込みがないと分かった節点を優先的に削除して枝刈りすることで，ZDD の幅を抑えるという工夫を試してみた．k を 5 から 25 の範囲で動かしてみた結果，多くの例題で ZDD 節点数を半分以下に抑えることができ，中には 4 分の 1 以下に削減できた例も見られるなど，一定の効果が認められた．ただし k を大きくするに従って先読みのための計算時間が増えるというトレードオフがあり，メモリが十分足りている場合は，何もしない方が数倍高速であるという結果となった．また，先読み枝刈りによる ZDD サイズ

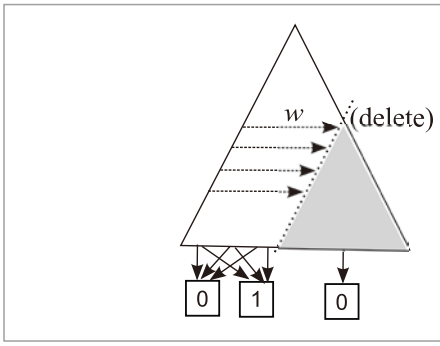

図 -6　幅に制限を加えた ZDD の枝刈り

削減効果は例題の性質に強く依存していて，k をかなり大きくしないと ZDD サイズが減らない例題もあった．

我々は，まず枝刈りなしのアルゴリズムを適用しても簡単に解ける例題を先に解いた上で，ZDD サイズが増大して簡単に解けない例題を自動的に抽出して先読み枝刈りを行うアルゴリズムを適用する，という戦略で 2 年目のコンテストに出場した．2 年目のコンテストでは，解を見つけ出す速度だけでなく，見つけた解の品質も評価されることになったため，ZDD により全解を列挙する手法にはある程度のアドバンテージもあると予想された．コンテストの結果，小規模〜中規模の例題では解集合の ZDD を素早く生成でき，その中から高品質の解を求めることができたが，一部の難しい例題にひっかかってしまって時間がかかり正答数を稼ぐことができず，1 年目のような上位進出はならなかった．ほかの参加チームの技術の完成度が 1 年目に比べて上がってきたということも要因として挙げられる．二度目の出場では，我々はナンバーリンク特有の性質はあまり考慮せず，ZDD 処理系としての汎用的な工夫を中心に考えていたことも成績が伸びなかった原因だったと考えているが，学生を中心とした研究活動の一環としてみれば，それはそれで致し方ないと考えている．

考察と今後の展望

以上で述べた通り，ZDD を用いたナンバーリンクの解法は，以下の点に弱みがある．

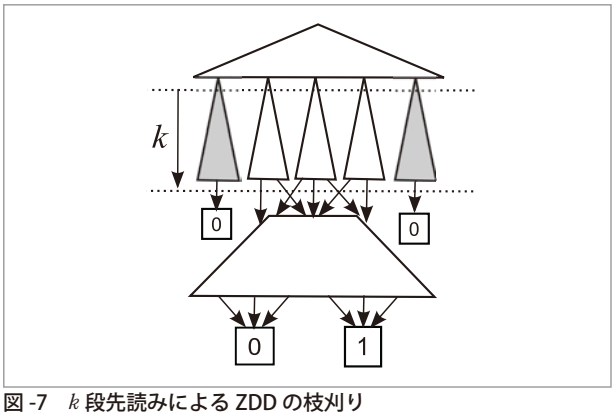

図 -7　k 段先読みによる ZDD の枝刈り

- ナンバーリンクの問題特有の制約条件をあまり深く活用できていない．
- 膨大な個数の解を圧縮して全列挙できるが，1 つの解に早くたどり着くことはそれほど強くない．

現実の LSI 配線問題では，計算機で扱える限界に近いサイズの大規模な例題も多くあり，最適解をあきらめて近似解を見つけるだけで精一杯で，多数の解をまとめて列挙するような余裕がないことが多いと思われる．しかし，たとえば再構成可能なアーキテクチャを設計する際に，再構成可能な配線のバリエーションを調べたり，どのような解でも必ず必要となるクリティカルなポイントを見つけたい，というような場合には ZDD を用いた列挙・索引化の技法が有用と思われる．

フロンティア法はナンバーリンクの問題に限らず，配電網や道路網，鉄道網，ガス，水道，通信網などの解析や制御，さらに避難所の配置や，選挙の区割り問題など，社会的に重要なさまざまな応用に深くかかわっている．我々の研究グループでは，Python をベースとした使いやすいインタフェースを装備したフロンティア法の処理系を「Graphillion」[3] という名前で公開している．興味のある方は一度ご覧いただければ幸いである．

参考文献

1) Knuth, D. E. : The Art of Computer Programming : Bitwise Tricks & Techniques ; Binary Decision Diagrams, Vol.4, fascicle 1. Addison-Wesley (2009).
2) 湊　真一：BDD/ZDD を用いたグラフ列挙索引化技法（特集 BDD/ZDD を用いた新しい列挙索引化技法（フロンティア法）とその応用），OR 学会誌，Vol.57, No.11, pp.597-603 (2012).
3) Inoue, T., et al. : Graphillion, http://graphillion.org/ (2013).

(2017 年 11 月 16 日受付)

謝　辞　アルゴリズムデザインコンテストのチームメンバの安田宜仁氏，鈴木浩史氏，岩下洋哲氏，中澤良男氏，孫浩氏に感謝いたします．本研究の一部は JST ERATO 湊離散構造処理系プロジェクト，および科研費 基盤 (S) 15H05711 の助成による．

■湊　真一（正会員）　minato@ist.hokudai.ac.jp

1988 年 京大・工・情報工学科卒業，1990 年 同大学院修士，1995 年 博士（社会人）修了．博士（工学）．1990 年 日本電信電話（株）入社．NTT 研究所にて大規模論理データ処理アルゴリズムの研究に従事．2004 年 北大・情報科学研究科助教授．2010 年より同教授．2009 〜 2015 年 JST ERATO 湊離散構造処理系プロジェクト研究総括（兼務）．BDD（二分決定グラフ）を用いた離散構造の処理に興味を持つ．著書 “Binary Decision Diagrams and Applications for VLSI CAD”（Kluwer，1995 年）．電子情報通信学会シニア会員，人工知能学会，IEEE 会員，本会シニア会員．