A-029

A*探索に基づく組合せ最適化問題の上位解列挙と ZDD の構築
Enumerating k-Best Solutions and ZDD Construction for Combinatorial Optimization Problems Based on A* Search

赤川 雄紀[1)]　川原 純[1)]　湊 真一[1)]
Yuki Akagawa　Jun Kawahara　Shin-ichi Minato

1　はじめに

多数のアイテムの中から条件を満たすようなアイテムの組合せを探索する問題は組合せ問題と総称され、実社会でも様々な場面で現れる。組合せ問題において最適解を求める問題を組合せ最適化問題と呼び、近年では最適解のみではなく、実行可能な解を全て列挙する研究も増えている。また、解の個数を制限する手段としてコスト関数を導入した、コスト制約組合せ問題を考えることができる。これにより、利用者はより容易に自分の求める結果を得ることができる。

組合せ問題の解集合はアイテム組合せの集合であるが、ZDD（ゼロサプレス型二分決定グラフ）[4] というデータ構造は組合せ集合を効率よく圧縮して解集合を保持できるため、ZDD に関する様々な研究が行われている。コスト制約組合せ問題の解を列挙する方法としては、ZDD の集合演算を用いて解集合 ZDD を計算する方法が研究されている [8]。この手法では、最終目的の ZDD を構築する途中で探索を中断すると、正しい上位解の集合が得られないことがあるという問題点がある。

そこで本稿では、あるタイミングで構築を中断してもその時点までの ZDD が正しい上位解を列挙できているような、コスト制約を満たす実行可能解集合 ZDD を構築する手法を提案する。従来のコスト制約つき列挙手法では、トップダウン方式で ZDD を構築する方法が研究されている [6][7]。しかしどのような手法でも、問題の規模が大きくなることでメモリの容量を超えてしまう。メモリ容量を超える前に途中で構築を中断する場合には、正しい上位解集合を得ることの保証ができなくなる。そこで本研究では、最適解から順に解が見つかることが知られている A*探索を用いて、途中で構築を中断しても常にその時点での正しい上位解を得られるような探索及び ZDD の構築手法を提案する。

2　準備

2.1　組合せ問題

n 個のアイテム集合 $I = \{1, 2, \ldots, n\}$ を定義し、I の任意の部分集合 $X \subseteq I$ を「アイテム組合せ」と呼ぶ。制約条件を表す集合関数 $f : 2^I \to \{0, 1\}$ が与えられたとき、$f(X) = 1$ を満たす実行可能解の集合は、

$$\mathcal{S}_f = \{X \subseteq I \mid f(X) = 1\}$$

と表せる。また、各アイテム $i (1 \le i \le n)$ について整数値のコスト $c_i \in \mathbf{Z}$ が定義されている時、アイテム組合せ X のコストを $\mathrm{Cost}(X) = \sum_{i \in X} c_i$ と定める。「組合せ最適化問題」とは $\mathrm{Cost}(X^*)$ が最小（または最大）になるような実行可能解 $X^* \in \mathcal{S}_f$ を探索する問題である。この組合せ最適化問題の派生として、アイテム組合せのコスト関数に閾値 $b \in \mathbf{Z}$ を与えることで、$\mathrm{Cost}(X) \le b$（または $\mathrm{Cost}(X) \ge b$）を満たす解 $X \in \mathcal{S}_f$ を求める組合せ問題を考えることができる。このような問題を「コスト制約組合せ問題」と呼ぶ。

2.2　0/1 ナップザック問題

本稿では、コスト制約組合せ問題の例としてナップザック問題を用い、その中でも基本的な 0/1 ナップザック問題を扱う。0/1 ナップザック問題では、各アイテム i を高々 1 回しか使用できないという制約がある。ナップザック問題では、アイテム集合 $I = \{1, 2, \ldots, n\}$ が与えられ、それぞれのアイテム i が重み $w_i > 0$ と価値 $p_i > 0$ を持っている。また、ナップザックの重さ制約 $c > 0$ も与えられる。また、I の任意の部分集合 $X \subseteq I$ について、$w(X), p(X)$ を以下のように定義する。

$$w(X) = \sum_{i \in X} w_i, p(X) = \sum_{i \in X} p_i.$$

ナップザック問題を解くということは、重さ制約 $w(X^*) \le c$ を満たし、$p(X^*)$ が最大となるアイテム組合せ X^* を見つけることである。

2.3　ゼロサプレス型二分決定グラフ

ZDD (Zero-suppressed Binary Decision Diagram; ゼロサプレス型二分決定グラフ)[4] は DAG (Directed acyclic graph; 有向非巡回グラフ) による組合せ集合の表現方法であり、BDD(Binary Decision Diagram; 二分決定グラフ)[1] の派生系である。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。葉の値が 1 の時はその組合せが集合に属することを表し、0 の時は属さないことを表す。また、1 の葉を 1-終端、0 の葉を 0-終端と呼ぶ。場合分け二分木に対して、BDD とは異なる以下の圧縮規則を用いて「既約」な形を得る。

1. 1-枝が 0-終端を直接指しているような節点を削除し、0-枝の行き先に直結させる。
2. 共通の行き先を持つ分岐節点が 2 個以上あれば 1 個にまとめて他を削除する。

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すことができる。

2.4　A*探索

A*探索は、グラフの 2 点間の最短距離を探索する技法である [2]。探索方法はダイクストラ法の拡張である。ダイクストラ法や A*探索ではグラフ上の各頂点 v について、現時点での評価値 $f(v)$ が与えられており、未探索の頂点のうち $f(v^*)$ が最も小さな頂点 v^* を次の探索頂点として探索済みにし、探索頂点に隣接している頂点の評価値を更新する。A* 探索では評価値は

1)　京都大学大学院情報学研究科 Graduate School of Informatics, Kyoto University

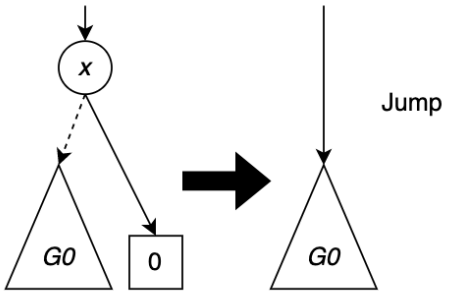

図 1　ZDD 圧縮規則 1

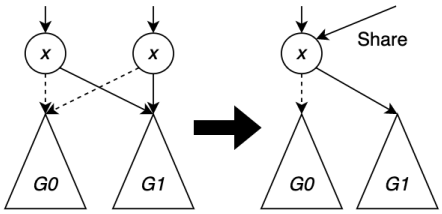

図 2　ZDD 圧縮規則 2

$f(\nu) = g(\nu) + h(\nu)$ と定義される。ここで、$g(\nu)$ は開始頂点から ν までの最短距離であり、$h(\nu)$ は ν から終了頂点までの最短距離の予測値である。$h(\nu) = 0$、すなわち、最短距離の予測値を利用せず、現時点での開始頂点から頂点 ν までの最短距離が最小となる ν を次の探索頂点とする方法がダイクストラ法に対応する。関数 $h(\nu)$ は ν から終了頂点までの真の最短距離 $h^*(\nu)$ を上回らない、即ち、任意の ν に対して、

$$0 \leq h(\nu) \leq h^*(\nu) \tag{1}$$

であるとき、ν は許容的であるといい、A*探索アルゴリズムが返す結果は最適（最小）となることが保証されている。A*探索を本稿で扱うような最大化問題に適用する場合は、任意の ν に対して、

$$0 \leq h^*(\nu) \leq h(\nu) \tag{2}$$

であるとき、結果は最適（最大）解が得られることが保証されている。

また、諸問題について A*探索を用いてコスト順に解を列挙する方法の研究も行われている [3][5]。A*探索で最適解を発見した後、探索を終了せずバックトラックを行うことで 2 番目に良い解を必ず発見することができる。これを繰り返すことで、評価値の順で全ての解を列挙することができる。

3　提案手法

今回提案する手法は、0/1 ナップザック問題の重さ制約を満たすアイテム組合せのうち、価値が大きい順に指定した個数のアイテム組合せの集合を表す ZDD を構築する。今回の提案手法は、0/1 ナップザック問題以外の組合せ最適化問題にも適用可能であると考える。初めに、価値が大きい順に指定した個数のアイテム組合せの集合を表す木構造を A*探索により作成し、その木構造を目的の ZDD に変換する。木構造はアイテム組合せの集合を非圧縮の状態で保持しているため、本アルゴリズムはアイテム組合せの個数に依存する計算時間と記憶量を必要とする。木構造を経由せずに直接 ZDD を構築する手法の提案は今後の課題である。

本節では、0/1 ナップザック問題の重さ制約を満たすアイテム組合せの集合を表す木構造を構築する手法を提案する。A*探索を用いることで、価値が大きい順に指定した個数のアイテム組合せの集合を含ませることができる。本稿ではこの個数を k で表す。

アルゴリズムが出力する木構造 T は以下の性質をもつ。T の葉は 0-終端または 1-終端である。T の非終端節点は $1, \ldots, n$ のいずれかのラベルと、0-枝、1-枝の 2 つの出る枝をもつ。ラベル i の節点の 0-枝（1-枝）はそれぞれアイテム i を解に含めない（含める）ことを意味する。ラベル i の節点の枝の先の節点は終端であるか、またはラベルが $i+1$ の節点である。T の根から 1-終端までの経路が 1 つのアイテム組合せに対応し、T の根から 1-終端までの経路集合が、T が表すアイテム組合せの集合となる。T は、節点を共有しない ZDD とみなすことができる。T に 2.3 節で説明した節点の共有ルールを可能な限り適用することで ZDD が得られる。

非終端節点 ν は $\nu = (i, d_0, d_1, H, f(\nu))$ で定義される。i はラベル、d_i $(i = 0, 1)$ は i-枝の先の節点、H は根から ν までの経路で 1-枝を選んだ箇所に対応するアイテムの集合（すなわち、これまでナップザックに含めたアイテムの集合）、$f(\nu)$ は以下で定義する A* 探索における ν の評価関数の値である。

A*探索では、T を根から終端まで「探索」すると考える。到達した終端の最初から k 個目までを 1-終端とし、残りの終端を 0-終端とすることで、T は k 個のアイテム組合せの集合を表すようになる。A*探索の評価関数の許容的な性質により、T は価値が上位 k 個のアイテム組合せを表す。

A*探索の予測関数 $h(\nu)$ は以下で定義される。

$$h(\nu) = \max_{i \in I \setminus H} \{p_i / w_i\} \cdot (c - w(H))$$

ここで、p_i/w_i とはアイテム i の価値効率を表している。$(c - w(H))$ は、ナップザックの残り重みを表す。したがって、$h(\nu)$ の意味は、ナップザックの残りの重みを、残りのアイテムのうち価値効率が最大のもので埋め尽くした場合に得られる価値の値である。$h(\nu)$ は許容的である。評価関数 $f(\nu)$ は、$f(\nu) = g(\nu) + h(\nu) = p(H) + \max_{i \in I \setminus H}\{p_i/w_i\} \cdot (c - w(H))$ である。

A*探索は以下の通り動作する。Q を空の優先度付きキューとし、Q から要素を取り出す際は $f(\nu)$ が最大になる節点 ν が取り出されるとする。根節点 $(1, \perp, \perp, \emptyset, 0)$ を Q に格納する。$\perp$ は、子節点は未確定という意味である。Q が空になるまで以下を繰り返す。Q から節点を取り出し、$\nu = (i, \perp, \perp, H, f(\nu))$ とする。

ν が非終端節点の場合を考える。最初に、ν の 0-枝の先を考える。これはアイテム i をナップザックに入れないことに相当する。節点 $\nu_0 = (i+1, \perp, \perp, H, f(\nu_0))$ を作成する。ここで、$f(\nu_0)$ は ν_0 の評価値を計算したものである。ν の d_0 として ν_0 を設定する。次に、ν の 1-枝の先を考える。これはアイテム i をナップザックに入れることに相当する。$w_i > c - w(H)$ であれば、アイテム i を入れられないことに相当するため、ν の 1-

枝の先を 0-終端とする。$w_i \leq c - w(H)$ であれば、節点 $(\nu_1) = (i+1, \bot, \bot, H \cup \{i\}, f(\nu_1))$ を作成する。ここで、$f(\nu_1)$ は ν_1 の評価値を計算したものである。ν の 1-枝の先を ν_1 にする。

Q から節点を取り出した際、$i = n+1$ すなわち最後のアイテムまで処理したか、または、$w(H) = c$ すなわち、ナップザックの残り容量が 0 のときは、その時点で有効なアイテム組合せが得られており、他のアイテムをナップザックに入れることはできないので、ν を 1-終端に変更する。これまで得られた解の個数を表すカウンタ `count` を 1 増やす。

カウンタ `count` が k になった時点で、キューに残る節点はすべて 0-終端に変換する。節点中の $\bot$ はすべて 0-終端に置き換える。

アルゴリズムの正当性の証明は省略するが、本手法で正しく木構造が出力されることを簡単に述べる。ある節点 $\nu = (i, d_0, d_1, H, f(\nu))$ について、アイテム組合せ H が解、すなわち価値が上位 k 個以内であるとする。木構造 T は H を保持していることを述べる。$w(H) = c$、すなわち、ナップザックの残り容量が 0 のとき、アルゴリズムの動作中で ν は 1-終端に変換される。したがって、T は H を保持している。$w(H) < c$、すなわち、ナップザックの残り容量が 0 より大きく、他の i 以上の（残りの）アイテムをナップザックに入れない場合、ν は終端にはならないが、ν の 0-枝の先を繰り返したどると、ラベルが $n+1$ の節点 ν' に到達する。ν' の評価値は $p(H)$ であり、ν' に到達するまでに現れる節点（途中節点と呼ぶ）の評価値は、h の許容的な性質によって $p(H)$ 以上である。したがって、H が価値上位 k 個以内であるなら、途中節点と ν' が、他の価値上位 k 個に入らない節点よりも先に優先度付きキューから取り出される。ν' は 1-終端に変換されるので、T は H を保持している。

4 実験結果

実験に用いた計算機は、プロセッサは 1.6 GHz デュアルコア Intel Core i5、メモリは 8 GB 2133 MHz LPDDR3 である。

4.1 使用したアイテムインスタンス

今回の実験では、[1 : 100] の乱数幅で生成した重みと価値をもつアイテム 20 個について実験を行った。ナップザックの容量は全アイテムの重み総和の 50%とした。

4.2 解の発見順序の検証

図 3 は、解の発見された順番とその評価値の関係を示しており、横軸が何番目に発見された解か、縦軸は解の評価値を表している。グラフは単調減少になっており、A*探索の特徴である、最適解から順番に見つかっていくことが確認できる。また、途中で中断されても常に正しい上位解集合を保持できることもわかる。

4.3 アイテム順序変更による A*探索性能の比較

ナップザック問題を貪欲に解こうとしたときには、価値効率 (p_i/w_i) が大きいアイテムから順番に選択していくことになる。逆に、価値効率が小さな値のアイテムから選択していくと最適解から遠のくことになる。今回の A*探索では、どのようなアイテム順序で探索を進めたら効率よく速く解が見つかるのかを考察するために、重み、価値、価値効率の 3 つの指標に関してソートして実験を行った。

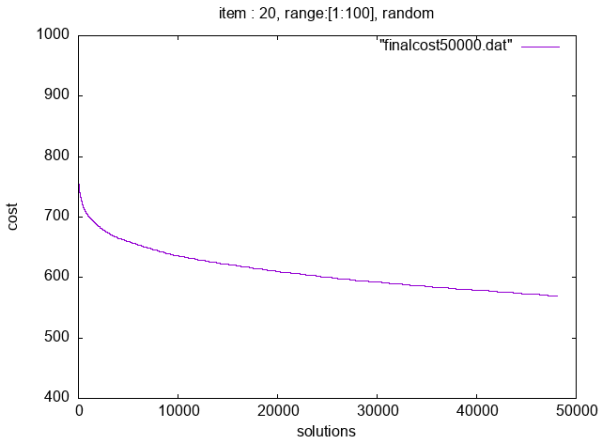

図 3 解の発見順とその評価値

図 4 は、３種類のアイテム順序に対して、ステップ数と見つけた解の個数を表したグラフである。上位解が見つかる速度に注目すると、重み順ソートの場合で解の発見が遅いことがわかる。一方で全ての解を見つける時には重み順ソートは最も速く、価値効率順ソートが最も遅いという結果になった。

4.4 優先度付きキューの要素数の変化

今回提案する A*探索では、優先度付きキューの要素数の推移は規則的である。推移の仕方は以下の 3 通りである。

- キューから取り出した節点 ν が解である（終了条件を満たしている）とき
  ν が取り出され、新たな節点は生成されないので、要素数は −1
- ν は解ではない（終了条件を満たしていない）が、１枝側の節点が枝刈りされるとき
  ν が取り出され、新たな節点が１つ生成されるので、要素数は $-1+1 = \pm 0$
- ν が解ではなく、０枝と１枝の両方の節点が生成されるとき
  ν が取り出され、新たな節点が２つ生成されるので、要素数は $-1+2 = +1$

以上の規則より、枝刈りのタイミングでは優先度付きキューの要素数は変化せず、解が見つかるタイミングで優先度付きキューの要素数は減少する、即ち、キューの要素数の増加が緩やかであるとき、速く解が見つかっていることがわかる。

図 5 は、アルゴリズムが進む過程で優先度付きキューの要素数がどのように変化していくかを表した図である。横軸がステップ数（アルゴリズム中の while 文を繰り返した回数）、縦軸が優先度付きキューに入っている要素の数である。水色の線が重み、緑の線が価値、紫の線が価値効率についてソートした結果を表している。1×10^6 ステップあたりまでは、価値効率順を表す紫の線が一番優先度付きキューの要素数が低い位置で推移している。このことから、ナップザック問題の上位解を見つける速さは、価値効率順ソートが一番早いことがわか

る。一方で、優先度付きキューの要素数が 0 になるタイミングに注目すると価値効率順ソートは一番遅く、重み順、価値順の方が速く探索が終了していることがわかる。これは、ナップザック問題の解を全探索する場合には、価値効率順ソートが優れているわけでは無いことを示している。

また、優先度付きキューの要素数と内部で保持している木構造のサイズは一致し、このグラフは実行中のメモリの使用量の推移と見ることもできる。

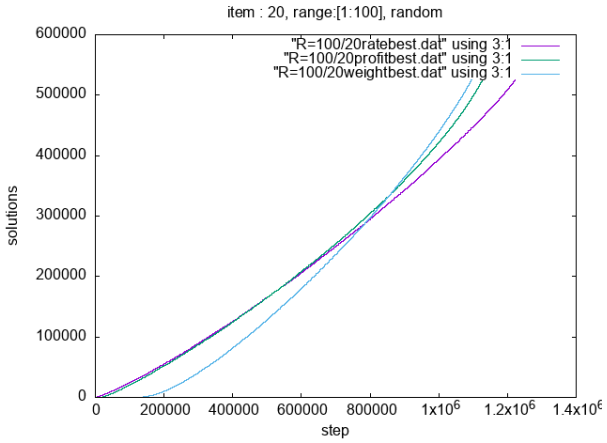

図 4　ステップ数と発見した解の個数の関係

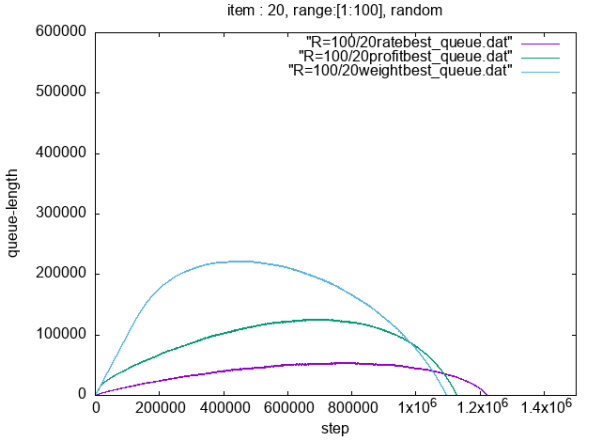

図 5　キューの要素数の推移

5　おわりに

今回の研究では、ナップザック問題のようなコスト制約組合せ問題に対して、A*探索に基づいて中間 ZDD を構成せずに上位解を列挙し、その解集合を表す ZDD を構築する方法を提案した。この方法では、A*探索が最適解から順に解を発見するという特性上、探索を中断しても正しい上位解を得られている。また、アイテム順序を固定して行う A*探索において、どのような指標でアイテムをソートしたら良いかを議論し、上位解の列挙という観点では、価値効率 (p_i/w_i) の降順に並べ替えるのが最も良いアイテム順序であることを実験的に求めた。

今後の課題としては、現在の実装では、アイテム集合から A*探索に基づき木構造を構築する第一段階と、木構造から ZDD を構築する第二段階に分かれているので、これらをひとつのアルゴリズムに組合せ、木構造を構築せずに直接 ZDD を構築する高速化が考えられる。ヒューリスティック関数については、現在用いているものでも最適解から順に良い解を出力する保証はあるが、より良いヒューリスティックを探すことでさらなる性能向上も望める。また新たに追加する機能として、メモ化技法を用いて ZDD の節点の展開をより効率の良いものにしたり、区間メモ技法 [8] を搭載することにより、より効率の良い枝刈り機能を追加できると考えられる。さらに、今回は上位解の指標として、1000 個など具体的な数字を用いて実験を行ったが、上位 5%の解を求めるなど、異なる形の制約への応用も今後の課題である。実社会への応用を想定し、多様な解を出力するような改良も考えたい。

謝辞

本研究の一部は、JSPS 科研費 JP20H00605、JP20H05794、JP20H05964 の助成を受けたものである。

参考文献

[1] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[3] E. L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management science*, 18(7):401–405, 1972.

[4] S. Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proceedings of the 30th International Design Automation Conference*, pages 272–277, 1993.

[5] K. G. Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations research*, 16(3):682–687, 1968.

[6] 安田宜仁, 西野正彬, and 湊真一. ZDD を用いた多次元ナップサック問題の厳密解法. In **人工知能学会全国大会論文集 第 *30* 回** *(2016)*, pages 1F33–1F33. 一般社団法人 人工知能学会, 2016.

[7] 戸田貴久, 斎藤寿樹, 岩下洋哲, 川原純, and 湊真一. ZDD と列挙問題—最新の技法とプログラミングツール. **コンピュータ ソフトウェア**, 34(3):3_97–3_120, 2017.

[8] 湊真一, 番原睦則, 堀山貴史, 川原純, 瀧川一学, and 山口勇太郎. ZDD の区間メモ化探索技法によるコスト制約組合せ問題の高速な解列挙. **研究報告アルゴリズム** *(AL)*, 2022(1):1–8, 2022.