3H-02

# 論理制約を伴う最短路問題を解く動的計画法の空間計算量の削減

竹内 文登 † 安田 宜仁 ‡ 西野 正彬 ‡ 湊 真一 †

† 北海道大学大学院 情報科学研究科 ‡ 日本電信電話（株）NTT コミュニケーション科学基礎研究所

## 1 はじめに

解に含まれるアイテムの組合せに制約があるナップサック問題などの重要な問題は，辺の取捨に関する制約を伴う有向非巡回グラフ（DAG）上の最短路問題に帰着できる．この問題を解く既存法に二分決定グラフを用いた動的計画法が知られている．しかし，既存法は探索空間中のすべての状態に対して経路復元のための情報を記憶するため，空間計算量が探索空間サイズに比例し，メモリ使用量が膨大になるという課題があった．本稿では最短路が経由する一つの頂点のみを求める問題は小さな空間計算量で計算できる点に着目し，再帰的に部分問題を解くことで最短路を求める手法を提案する．制約付きナップサック問題や制約付き配列アラインメント問題を解く場合，提案法の時間計算量は従来法と変わらない．実験では提案法のメモリ使用量が既存法の平均 1/100 以下となる一方，速度低下は平均 2 倍以内に抑えられることを確認した．

## 2 論理制約を伴う DAG 最短路問題

論理制約を伴う DAG 最短路問題は，入力として重み付きの DAG $G=(V,E)$ と，頂点 $s,t\in V$，辺に関する重み関数 $w:E\mapsto\mathbb{R}$，辺に関する制約を表現する論理関数 $F:\{0,1\}^{|E|}\mapsto\{0,1\}$ が与えられたとき，$F(\chi(p))=1$ となるような経路 $p\in 2^E$ のうち，辺重みが最小となる $s$-$t$ 経路を求める問題である．ただし特性関数 $\chi:2^E\to\{0,1\}^{|E|}$ は $e_i\in p$ ならば $\chi(p)$ の $i$ 番目に要素を 1 とし，そうでないならば 0 とする関数とする．以下では辺 $(u,v)$ を $e_{uv}$ と表記する．図 1 に例題を示す．この例では，論理制約 $F=(e_{v_{11}v_{12}}\wedge e_{v_{33}v_{44}})\vee e_{v_{22}v_{33}}$ を満たす $v_{11}$ から $v_{55}$ までの最短路を求める．

同時に取ることのできるアイテムに制約のあるナップサック問題や，対応させる要素間に制約を伴う配列アラインメントといった種々の論理制約を伴う組合せ最適化問題は，この制約を伴う DAG 最短路問題に帰着することができる．

## 3 BDD-Constrained Search（BCS）

論理制約を伴う DAG 最短路問題を解く従来手法に BDD-Constrained Search（BCS）[3] がある．この手法では二分決定グラフ（BDD）[1] と呼ばれるデータ構造を用いて論理制約を表現する．BDD は論理関数 $F:\{0,1\}^n\to\{0,1\}$ を図 1 の右のようにグラフの形で

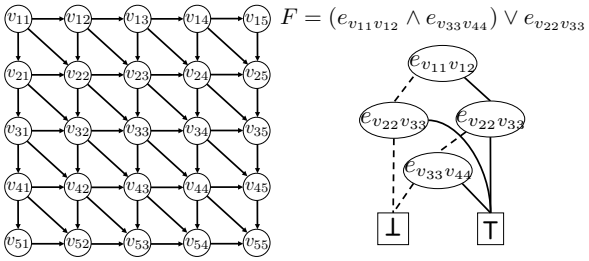


図 1: 論理制約を伴う DAG 最短路問題の例

一意に表現するものであり，この例では $F=(e_{v_{11}v_{12}}\wedge e_{v_{33}v_{44}})\vee e_{v_{22}v_{33}}$ を表現している．BDD 中の各節点の子孫からなる部分グラフも BDD であり，もとの BDD が表現する論理関数の部分関数を表現する．ルート節点から 0 枝（破線）または 1 枝（実線）を辿る操作は，節点に書かれたラベル変数に 0 または 1 を代入することに対応し，行き着いた頂点以下の部分的な BDD がその変数割り当てに対する部分関数に対応する．

BCS はよく知られた DAG 最短路問題を解く動的計画法を拡張したものである．具体的にはある頂点に至るまでの経路のうち，残りの辺に関する（部分的な）制約が等しい最短経路を記憶する動的計画法によって，制約付き最短路問題を解く．たとえば図 1 の例において，経路 $v_{11}\to v_{22}$ と経路 $v_{11}\to v_{21}\to v_{22}$ では，残りの辺に関する制約がどちらも $F'=e_{v_{22}v_{33}}$ と等しいため，経路長がより短い経路のみを記憶する．一方，経路 $v_{11}\to v_{12}\to v_{22}$ は，残りの辺に関する部分的な論理制約が $F''=e_{v_{22}v_{33}}\vee e_{v_{33}v_{44}}$ であり，先の 2 つの経路とは残りの制約が異なるためどちらの経路も記憶をする．BCS では，部分制約を BDD の各節点で表現し，DAG 上の頂点 $v$ と残りの部分制約を表現する BDD の節点 $n$ のペア $(v,n)$ を一つの状態とする探索空間で動的計画法を実行することで上記の探索を実現する．以下ではこのペアを探索状態，または状態と呼ぶ．BCS では状態 $(v,n)$ で経路を記憶する際，状態 $(v,n)$ に至る一つ前の状態のみを記憶し，ゴール状態からスタート状態に至るまでバックトラックにより経路を復元する．

BCS の時間計算量は $O(|E|W)$，空間計算量は $O(|V|W)$ である．ここで $W$ は BDD において同じ変数に対応する節点の数の最大値と定義する．空間計算量は，状態数が $O(|V|W)$ であり，かつ各状態についてその状態に至る最短経路の経路長と最短路での一つ前の状態を記憶する必要があることに由来する．

## 4 提案法

本節では空間計算量を削減した手法を提案する．提案法では，最短路が経由する一つの頂点のみを求める

---

**A Space Efficient Dynamic Programming Algorithm for Logically-Constrained Shortest Path Problems**

Fumito TAKEUCHI†, Norihito YASUDA‡, Masaaki NISHINO‡, Shin-ichi MINATO†

†Graduate School of Information Science and Technology, Hokkaido University

‡NTT Communication Science Laboratories, NTT Corp.

**Algorithm 1** $MemSavingBCS(G, B, (v_s, n_s), (v_t, n_t))$

入力： DAG $G = (V, E)$, BDD $B$，スタート状態 $(v_s, n_s)$，ゴール状態 $(v_t, n_t)$（$v_s, v_t \in V$，$n_s, n_t$ は BDD の節点）
出力： 論理制約を満たす $G$ 上の $v_s$-$v_t$ 最短路

```
1: Cost[v_s][n_s] ← 0, Mid[v_s][n_s] ← (v_s, n_s), p ← []
2: for all vertex u in topological order from v_s to v_t do
3:   for all state (u, n) do
4:     for all edge e_uv outgoing from u do
5:       n' ← followBDD(e_uv, n)
6:       if n' = ⊥ then continue
7:       if Cost[v][n'] > Cost[u][n] + w_uv then
8:         Cost[v][n'] ← Cost[u][n] + w_uv
9:         Mid[v][n'] ← Mid[u][n]
10:        if isMidState((v, n')) then
11:          Mid[v][n'] ← (v, n')
12:    delete Cost[u][n] and Mid[u][n]
13: (v_m, n_m) ← Mid[v_t][n_t]
14: if (v_m, n_m) = (v_s, n_s) then
15:   return {e_{v_s v_t}}
16: add MemSavingBCS(G, B, (v_s, n_s), (v_m, n_m)) to p
17: add MemSavingBCS(G, B, (v_m, n_m), (v_t, n_t)) to p
18: return p
```

問題は小さな空間計算量で計算できることを利用し，この問題を再帰的に解くことで最短路を求める．最短路が経由する一つの頂点を求める問題は，経路長を記憶するのと同時に，経由した頂点を一つ記憶することで解くことができる．また頂点を一つ求める問題では最短路を復元する必要がないため，全ての状態において一つ前の状態を記憶する必要がなくなる．そのため探索の終わった状態をメモリから削除することができ，空間計算量を削減することができる．類似の従来法にナップサック問題や配列アラインメントを小さな空間計算量で解く手法がある [2, 4]．

提案法のアルゴリズムを Algorithm 1 に示す．1-13 行目で $v_s$ から $v_t$ への制約を満たす最短路長と最短路が経由する状態を求め，16，17 行目でスタート状態から経由状態までの部分問題と経由状態からゴール状態までの部分問題を再帰的に解く．この再帰を部分問題が自明な解を持つまで繰り返す（14，15 行目）．$Cost[u][n]$ は状態 $(u, n)$ までの最短路長を記憶し，$Mid[u][n]$ は $(u, n)$ までの最短路が経由する状態をひとつ記憶する．空間計算量を削減するため，状態 $(u, n)$ までの情報がそれ以降の探索で必要なくなった時点で，$Cost[u][n]$，$Mid[u][n]$ をメモリから削除する（12 行目）．関数 followBDD$(e, n)$ は BDD 節点 $n$ から辺 $e$ を通ったときの部分関数を表現する BDD 節点を返す関数である．isMidState$(s)$ は，状態 $s$ が事前に定められた中間状態集合 $S$ に含まれている場合に真を返す関数である．ただし，$S$ は以下の二つの要件を満たすように設計する必要がある．(1)$S$ は元問題の最短路がどのような経路を辿ったとしても，$S$ 中の 1 つ以上の状態を経由するような集合である．(2) 状態 $s_1$ から状態 $s_2$ までの最短路の計算にかかるステップ数を $K(s_1, s_2)$ としたとき，$S$ に含まれるすべての状態 $s$ に対して，$K((v_s, n_s), s) + K(s, (v_t, n_t)) \leq r \times K((v_s, n_s), (v_t, n_t))$ を満たす 1 未満の定数 $r$ が存在する．

提案法の時間計算量は従来法と等しい．なぜなら，各再帰の深さにおいてステップ数が再帰前のステップ数の高々1 倍未満に減ることから，$d$ を再帰の深さ，$r$ を 1 未満の定数とすると，全体のステップ数が

$$|E|W + \frac{|E|W}{r} + ... + \frac{|E|W}{r^d} = O(|E|W) \quad (1)$$

で抑えられるためである．図 1 の例では，たとえば最短路が $v_{33}$ を通ることが分かったとき，$v_{11}$-$v_{33}$ 最短路問題と $v_{33}$-$v_{55}$ 最短路問題では，最短路が $v_{14}, v_{24}, v_{34}$ や $v_{41}, v_{42}, v_{43}$ などの頂点を経由することがないため，これらの頂点への辺を探索する必要がなくなる．この性質は再帰的に成立するため (1) 式が成立する．提案法の空間計算量は，トポロジカル順序における頂点分離数（vertex separation number）を $S$ としたとき $O(SW)$ となる．

| | 従来法：BCS | | 提案法 | |
|---|---|---|---|---|
| 容量 | メモリ MB | 時間 秒 | メモリ MB | 時間 秒 |
| 100 | 42 | 0.045 | 2 | 0.045 |
| 200 | 132 | 0.210 | 2 | 0.175 |
| 500 | 266 | 0.415 | 4 | 0.495 |
| 1,000 | 886 | 1.810 | 9 | 1.810 |
| 2,000 | 2,544 | 5.605 | 30 | 7.055 |
| 5,000 | 7,098 | 15.174 | 70 | 18.965 |
| 10,000 | 12,230 | 26.533 | 126 | 34.633 |
| 20,000 | 17,089 | 34.646 | 158 | 48.682 |
| 50,000 | 21,533 | 36.279 | 181 | 62.501 |

表 1: メモリ使用量（MB）と計算時間（秒）の比較

## 5 実験

提案法のメモリ使用量と計算時間を計測し従来法と比較を行った．実験では，ある二つのアイテムを同時に選択できない制約を伴うナップサック問題を解いた．具体的な設定は以下の通り．アイテム数は 1,000，アイテムの価値，重さは [1,100] のランダム，排他制約の数はランダムな 10 ペアとし，ナップサックの容量は表 1 に示すように 100 から 50000 の間で変え，各設定において 100 個の問題を生成し合計 900 問を解いた．実験は Intel Core i7 3.5GHz，メモリ 32GB の計算機で行った．

従来法は 900 問中 91 問をメモリ不足のため求解できなかったが，提案法はすべての問題を解くことができ，より大規模な問題を解くことができた．従来法が求解できた問題での使用メモリ量と計算時間の平均値を表 1 に示す．すべての問題で提案法のメモリ使用量が既存法より小さく平均 1/100 以下であった．速度低下は平均 2 倍以内であった．

**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

[1] Sheldon B. Akers. Binary decision diagrams. *Computers, IEEE Trans. on*, 100(6):509–516, 1978.

[2] Daniel S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, 1975.

[3] Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. BDD-constrained search: A unified approach to constrained shortest path problems. In *Proc. of AAAI*, pages 1219–1225, 2015.

[4] Ulrich Pferschy. Dynamic programming revisited: Improving knapsack algorithms. *Computing*, 63(4):419–430, 1999.