制約付きナップサック問題に対するBDD-Constrained Searchの省メモリ化

竹内文登[1]、安田宜仁[2]、西野正彬[2]、湊真一[1]（北大[1]、NTT[2]）

本発表の概要

- BDD-Constrained Search(BCS) [1] は「論理制約を伴う組合せ最適化問題」を解く
- 実験的にBCSは空間計算量がネックでスケーラビリティが低かった
- 提案法はBCSと同じ時間計算量で空間計算量を削減
- 実験より、2倍を超えない計算時間で大幅な省メモリ化に成功した
- これまでBCSでは解けなかったサイズの問題も解けるようになった

制約付きナップサック問題

- 入力
  - N個のアイテム（価値v_i、重さw_i）
  - ナップサックの容量 C
  - 論理関数 $f(x)$
- 出力
  - 価値v_iの和が最大となるアイテムの組合せ x
  - ただし、重さw_iの和が C を超えない
  - さらに、$f(x)=1$

$$\text{Maximize}: \sum_{i=1}^{N} x_i v_i$$

$$\text{s.t.} \sum_{i=1}^{N} x_i w_i \le C$$

$$f(x)=1$$

$$x \in \{0,1\}^N$$

BDD-Constrained Search (BCS) [1] とは？

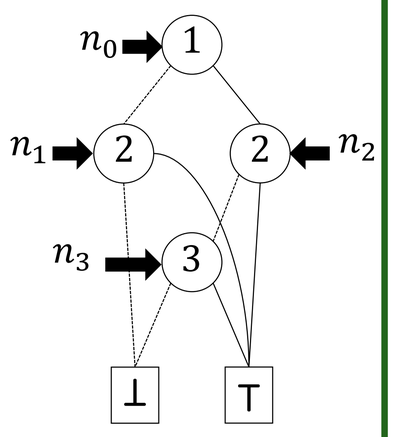
- 対象とする問題：「論理制約を伴う組合せ最適化」
  - ナップサック問題
  - 編集距離計算
  - ビタビ経路探索問題
  - 最長共通部分列問題 etc.

＋　論理制約

- 入力
  - 動的計画法により得られるDAG（有向非巡回グラフ）
  - 論理制約を表現するBinary decision diagram (BDD)

価値	重さ
-	-
3	2
4	2
6	3

N,W	0	1	2	3	4	5
-	0					
1	0		3			
2	0		4		7	
3	0		4	6	7	10



- アルゴリズム
  - DAG上の各頂点を幅優先的に操作
    - 各頂点で（以降の論理制約、その頂点までの最短路長）のペアを記憶
    - 「以降の論理制約」が等しいペアが存在する場合は、「その頂点までの最短路長」が小さいペアのみを記憶
    - **経路復元のため、更新があるときは前のペア状態も記憶**
  - バックトラックで経路を復元し最短路を得る

価値	重さ
-	-
3	2
4	3
6	5

N,W	0	1	2	3	4	5
-	$(n_0,0)$					
1	$(n_1,0)$		$(n_2,3)$			
2	$(\bot,0)$		$(n_3,3)$ $(\top,4)$		$(\top,7)$	
3			$(\bot,3)$ $(\top,4)$		$(\top,7)$	$(\top,10)$

- **時間計算量：$O(E\times W)$**（E：DAGの辺数、W：BDDの幅）
- **空間計算量：$O(E\times W)$**
  - ナップサック問題の場合は、時間、空間計算量ともに $O(N\times C\times W)$

ナップサック問題に対する省メモリDP解法 [2]

- 省メモリ化の基本的なアイディア
  - DPの遷移に必要な**前の行の情報以外を忘れる**（スコア計算のみ可能）
- 経路復元のためのアルゴリズム
  - 最適経路が通る中間頂点を求める
  - 始点から中間頂点に対応するナップサック問題を再帰的に解く
  - 中間頂点から終点に対応するナップサック問題を再帰的に解く
  - 結果をマージ、解を出力

価値	重さ
-	-
3	2
4	3
6	5
4	2
4	4

N,W	0	1	2	3	4	5	6	7	8	9	10
-											
1											
2											
3											
4	0		4	4	7	8		11	10	13	14
5	0		4	4	7	8	8	11	10	13	14

- 時間計算量
  - $NC+\frac{NC}{2}+\frac{NC}{4}+\ \ldots+\ \frac{NC}{2^{\log N}}=O(N\times C)$
- 空間計算量
  - 常に２行しか持たないので$O(C)$

制約付きナップサック問題に対する省メモリBCS

- 制約ナップサック問題に対するBCSに、[2]のアイディアを利用
  - 「中間地点のペアを求める」を再帰的に解く
  - 各行では$O(C\times W)$個の要素を記憶
- 時間計算量
  - $NCW+\frac{NCW}{2}+\frac{NCW}{4}+\ \ldots+\ \frac{NCW}{2^{\log N}}=\boldsymbol{O(N\times C\times W)}$
- 空間計算量
  - 常に２行しか持たないので$\boldsymbol{O(C\times W)}$
- BCSと同じ時間計算量でアイテム数に依存しない空間計算量を実現

実験

- 目的：BCSと提案法の使用メモリと計算時間の比較
- アイテム数：1000
  - 価値、重さ：[1,100] ランダム
  - 制約：排他制約（2つのアイテムを同時に含んではいけない）10ペア
  - ナップサック容量：100,200,500,1000,2000,5000,10000,20000,50000
- 実験環境：intel Core i5、メモリ32G（Memory Out=30G）

	BCS			省メモリ版BCS		
容量C	メモリ(KB)	時間(s)	#MemOut	メモリ(KB)	時間(s)	#MemOut
100	42,048	0.045	0	1,740	0.045	0
200	131,938	0.210	0	2,446	0.175	0
500	266,028	0.415	0	3,512	0.495	0
1,000	885,800	1.810	0	9,452	1.810	0
2,000	2,544,116	5.605	0	29,648	7.055	0
5,000	7,098,258	15.174	1	70,408	18.965	0
10,000	12,229,797	26.533	5	126,388	34.633	0
20,000	17,089,428	34.646	28	157,593	48.682	0
50,000	21,533,188	36.279	57	181,317	62.501	0

参考文献

- [1] Nishino, M.; Yasuda, N.; Minato, S.; and Nagata, BDD-constrained search: A unified approach to constrained shortest path problems. 2015, In *Proc. of AAAI*, 1219–1225.
- [2] Pferschy, U. Dynamic Programming Revisited: Improving Knapsack Algorithms. 1999. In Journal of Computing, vol. 63, Issue 4, 419–430
- [3] Hirschberg, D. S. A Linear Space Algorithm for Computing Maximal Common Subsequences. 1975. In Journal of Commun. ACM, vol. 18, 341–343