数理解析研究所講究録
第1726巻 2011年 139-154

排他制約付きナップサック問題における上界の計算法およびその有効性

山崎 洋祐[1], スキアボーニ リッカルド[2],
イオリ マヌエル[3], 柳浦 睦憲[1], マルテッロ シルバノ[2]
Yosuke Yamasaki, Riccardo Schiavoni,
Manuel Iori, Mutsunori Yagiura, Silvano Martello

[1] 名古屋大学大学院情報科学研究科
[2]University of Bologna
[3]University of Modena and Reggio Emilia

概要

ナップサック問題 (knapsack problem) は情報科学の分野における基礎的な問題で, これまでに多くの研究がある. また, ナップサック問題に制約を加えた問題も多数存在する. 本論文ではそのひとつである排他制的付きナップサック問題 (disjunctively constrained knapsack problem) を対象とする. これは, 同時に選択できないアイテムに関する制約 (排他制約) と容量制約を満たすようにいくつかのアイテムを選択するとき, 選択したアイテムの価値の合計を最大化する問題である. これに対し, クリークを用いた上界値計算法を提案する. 本計算法により大規模な問題例や辺密度の高い問題例に対して高速に上界値を計算できることを計算実験により確認した.

1 序章

ナップサック問題 (knapsack problem)[1, 5, 10, 11, 13] は情報科学の分野における基礎的な問題で, これまでに多くの研究がある. また, ナップサック問題に制約を加えた問題も多数存在する. 本論文ではその中のひとつである排他制的付きナップサック問題 (disjunctively constrained knapsack problem)[2, 3, 14] を対象とする. この問題は, ナップサック問題や最大独立集合問題 (maximum independent set problem) を特殊ケースとして含む. よって排他制約付きナップサック問題は NP 困難である.

排他制約付きナップサック問題は, トラックによる配送計画などを考える際, 混載できない荷物の組合せがあるときにどのように荷物を積載すれば良いかを考える場合などに応用することができる. この他にも従業員の能力に基づいた人事管理, 予算内の物資の購入など多くの応用例の基礎となっている.

排他制約付きナップサック問題は入力として, 容器の容量 $c(\geq 0)$ と, アイテム集合 $V = \{1, \ldots, n\}$ の各アイテム $i \in V$ に対する重み $w_i(\geq 0)$ と価値 $p_i(> 0)$, および同時に選択できないアイテム対の集合 E が与えられる. 出力はアイテムの部分集合 $S \subseteq V$ であり, その部分集合 S に含まれるアイテムのどの対も E に含まれてはならない. また, 部分集合 S に含まれる各アイテムの重みの和は, 容量を超えてはならない. この条件下で部分集合 S に含まれるアイテムの価値の和を最大にするのがこの問題の目的である. 便宜上アイテムの添え字は単位重みあたりの価値 p_i/w_i が高い順に整列されているとする (すなわち $p_1/w_1 \geq p_2/w_2 \geq \cdots \geq p_n/w_n$). ただし, $w_i = 0$ のアイテムが存在する場合は, それらを先頭から p_i の降順に並べたのち, $w_i > 0$ なるアイテムを p_i/w_i の降順に並べた列の順序を用いる. また, 以下では $m = |E|$ とする. この問題に対する既存研究としては, T. Yamada らや Hifi と Michrafy による分枝限定法に基づく厳密解法の研

究 [2, 14] や，Hifi と Michrafy による発見的解法の研究 [3] などがある．

本研究では，排他制約付きナップサック問題に対する効率的な上界値計算法を提案する．アイテム集合 V を頂点集合，排他制約を表す集合 E を辺集合とするグラフを考える．頂点の部分集合で，任意の頂点間に辺が存在するものをクリークと呼ぶ．排他制約付きナップサック問題の任意の実行可能解 S は，任意のクリーク $C \subseteq V$ に対して C に含まれる頂点を高々1つしか S に含むことができないという条件を満たすが，上界値計算において線形計画緩和問題を考える際に，排他制約をこのようなクリークに基づく制約に置き換えることにより上界の改善が期待できる．しかし，クリークの中でもとくに極大なものだけを列挙する効率的なアルゴリズムは存在するものの [7]，クリークの個数は指数オーダーになり得るため，全てを列挙するのは問題例が大規模な場合などに現実的でない．また，分枝限定法において問題の上界を計算する際に一般的に用いられてきた線形計画緩和は，その計算を行う際に線形計画問題 (linear programming problem，LP 問題) に対する汎用解法 (以下，LP ソルバーと記す) を用いるため，大きな計算時間を要する．提案する手法は，特定の条件を満たすクリーク集合を生成することで，上界を計算する際に特殊なアルゴリズムを利用できるため，LP ソルバーを用いる必要がなく，高速な計算が可能である．

計算実験の結果，既存のアルゴリズムよりも非常に高速に上界値の計算が可能であることが確認できた．上界値については，辺密度の低い問題例に対してはあまり大きなクリークが生成されず既存のアルゴリズムの方が良い上界を与えたが，辺密度の高い問題例に対しては既存のアルゴリズムよりも良い上界を与えることができた．また，提案する上界値計算法を分枝限定法 [4, 6, 9, 12] に取り入れたところ，辺密度の高い問題例に対しては既存のアルゴリズムより高速に最適解を計算できることが確認できた．

2 辺制約による定式化とクリーク制約による定式化

本節では，排他制約付きナップサック問題の2つの定式化を考える．まず，排他制約を表す集合 E に基づく定式化を説明する．各アイテム i が解 S に含まれるならば 1，含まれないならば 0 の値をとる変数 x_i を用いて，排他制約付きナップサック問題を以下の問題 P のように定式化することができる:

$$P \quad \text{maximize} \quad z(x) = \sum_{i=1}^{n} p_i x_i, \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq c, \tag{2}$$

$$x_i + x_j \leq 1, \quad \forall (i,j) \in E, \tag{3}$$

$$x_i \in \{0,1\}, \quad i = 1, \ldots, n. \tag{4}$$

以下では，この定式化を辺制約による定式化と呼ぶ．

次に，クリーク制約に基づく定式化を説明する．前節で述べたように，実行可能解 S は任意のクリークから高々1つしかアイテムを選ぶことができない．この性質を利用してクリークに基づく定式化を考える．便宜上，注目するクリークに適宜番号をつけて各クリークを $C_k \subseteq V$ と記すことにし，あるクリーク集合 $Q = \{C_1, \ldots, C_r\}$ に対する以下の整数計画問題を考える:

$$P(Q) \quad \text{maximize} \quad \sum_{i=1}^{n} p_i x_i \tag{5}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq c, \tag{6}$$

$$\sum_{i \in C_k} x_i \leq 1, \quad \forall C_k \in Q, \tag{7}$$

$$x_i \in \{0,1\}, \quad i = 1, \ldots, n. \tag{8}$$

以下では，この定式化をクリーク制約による定式化と呼ぶ．

ここで，元の問題 P とクリーク集合 Q による問題 $P(Q)$ の関係を考える．まず，$Q = \{\{i,j\} \mid (i,j) \in E\}$ である場合には，制約 (2.3) と (2.7) が一致するので，問題 P と問題 $P(Q)$ は全く同じ問題となる．次に，Q がグラフ (V,E) 上の全てのクリークの集合である場合には，問題 P と問題 $P(Q)$ は等価となる．また，Q が全てのクリークの集合でなくても，Q が

$$\forall (i,j) \in E, \ \exists C_k \in Q, \ \{i,j\} \subseteq C_k \tag{9}$$

をみたすならば問題 P と問題 $P(Q)$ は等価となる．さらに，Q が (2.9) を満たすならば，

[問題 P の最適値]
≤[問題 $P(Q)$ の線形計画緩和問題の最適値]
≤[問題 P の線形計画緩和問題の最適値]

となり，クリークの導入により上界の改善が期待できる．

Q が任意に与えられた場合でも，問題 P の任意の実行可能解は問題 $P(Q)$ の実行可能解となる．つまり，問題 $P(Q)$ は問題 P の緩和問題となるので，任意の Q に対して問題 $P(Q)$ の最適値は問題 P の上界を与える．従って，問題 $P(Q)$ の線形計画緩和問題の最適値も問題 P の上界を与える．しかし，Q が任意に与えられた場合には，問題 $P(Q)$ の線形計画緩和問題の最適値と問題 P の線形計画緩和問題の最適値のうちどちらによる上界の方が良いかは問題例や Q のとり方に依存する．

本研究では上界を与える方法として双対問題を用いる．双対問題の任意の実行可能解の目的関数値は主問題 (最大化問題) の最適値の上界を与えるので，

[問題 $P(Q)$ の線形計画緩和問題の最適値]
≤[問題 $P(Q)$ の線形計画緩和問題の双対問題の任意の実行可能解の目的関数値]

が成り立つ．以上より

[問題 P の最適値]
≤[問題 $P(Q)$ の線形計画緩和問題の双対問題の任意の実行可能解の目的関数値]

となるので，任意のクリーク集合 Q に対し，問題 $P(Q)$ の線形計画緩和問題の双対問題の任意の実行可能解の目的関数値は元の問題 P の上界を与える．

3 クリーク分割による上界値計算法

本節ではクリーク分割に基づく上界値計算法を提案する．クリーク制約による定式化 $P(Q)$ において，Q がアイテム集合 V の分割になっている場合，その線形計画緩和問題の双対問題が特殊な構造を持ち，その最適解を容易に計算できる．本節ではまずこの性質を示したのち，このアイデアに基づく上界値計算法を提案する．

3.1 クリーク分割に対する双対問題

問題 $P(Q)$ において，クリーク集合 $Q = \{C_1, \ldots, C_r\}$ がアイテム集合 V の分割である場合を考える．すなわち Q は

- どのアイテムもひとつ以上のクリークに属する ($\forall i \in V,\ \exists C_k \in Q,\ i \in C_k$)
- 相異なるクリーク対は互いに疎である ($\forall C_k \neq C_l \in Q,\ C_k \cap C_l = \emptyset$)

という 2 つの条件を満たす．このようなクリーク分割 Q に基づく問題 $P(Q)$ の線形計画緩和問題に対する双対問題は以下のように定式化することができる:

$DCP(Q)$

$$\text{minimize} \quad f(\lambda, y) = c\lambda + \sum_{k=1}^{r} y_k \tag{10}$$

$$\text{subject to} \quad w_i\lambda + y_k \geq p_i,\ k : i \in C_k,\ \forall i \in V, \tag{11}$$

$$\lambda \geq 0,\ y_k \geq 0,\ \forall k : C_k \in Q. \tag{12}$$

この定式化において，λ をある値 $\hat{\lambda}$ に固定すると，各 y_k は互いに独立した制約しか持たない．よって，各 y_k を制約を満たした上で独立に最小化することで問題 $DCP(Q)$ の $\lambda = \hat{\lambda}$ の場合の最適解を $O(n)$ 時間で得ることができる．具体的には，各 $k = 1, \ldots, r$ に対し $y_k = \max\{0, \max_{i \in C_k}(p_i - w_i\hat{\lambda})\}$ となる．この y_k の値を $\hat{\lambda}$ の関数ととらえ，$y_k(\hat{\lambda})$ と記すことにする．このように考えると問題 $DCP(Q)$ の目的関数 $f(\lambda, y)$ を $c\lambda + \sum_{k=1}^{r} y_k(\lambda)$ のように λ の関数として考えることができる．便宜上これを $f(\lambda)$ と記す．また，$y_k(\lambda)$ は λ に関する線形関数の最大値からなる関数なので，$y_k(\lambda)$ が凸な区分線形関数であることは明らかである．関数 $y_k(\lambda)$ の一例を図 1 に示す．$f(\lambda)$ はこれらの k に対する和であるから，問題 $DCP(Q)$ の目的関数は変数 λ に対して下に凸な区分線形関数であることがわかる．

λ の探索には 2 分探索法を用いた．その際の λ の探索範囲について考える．まず，簡単のため，全ての i に対して $w_i > 0$ であると仮定する．このとき，単位重さあたりの価値の最大値 p_1/w_1 に対し，$\lambda \geq p_1/w_1$ の範囲では全ての $k = 1, \ldots, r$ に対して $y_k(\lambda) = 0$ となるため，問題 $DCP(Q)$ の目的関数 $f(\lambda)$ は $c\lambda$ となる．問題 $DCP(Q)$ は最小化問題であるが，仮定 $c \geq 0$ より $\lambda \geq p_1/w_1$ の範囲では $f(\lambda)$ は単調非減少となるため，この区間では $\lambda = p_1/w_1$ のときに得られる解が最良である．

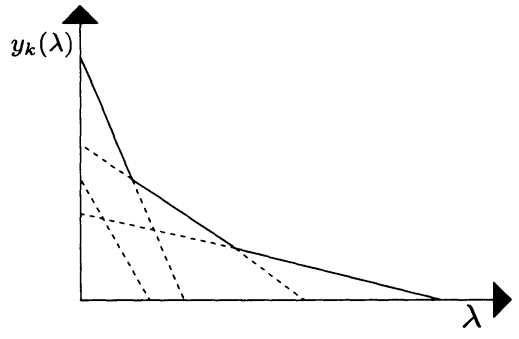

図 1. 関数 $y_k(\lambda)$ の例

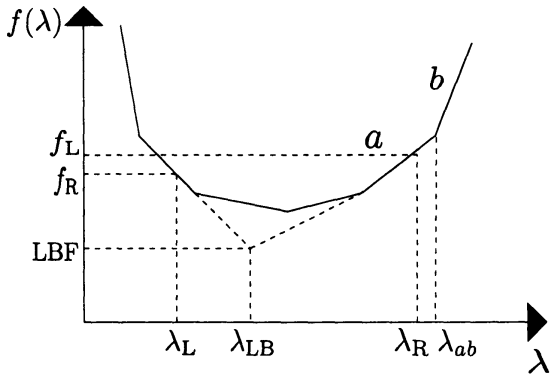

図 2. $f(\lambda)$ の下界値の計算例

$w_i = 0$ であるアイテムが存在する場合についても，$\lambda \geq \max\{p_i/w_i \mid w_i > 0\}$ の範囲で (全ての i について $w_i = 0$ のときは $\lambda \geq 0$ の範囲で) $f(\lambda)$ が単調非減少であることを同様に示すことができる．よって λ の探索範囲は 0 から $\max\{p_i/w_i \mid w_i > 0\}$ までで十分である (全ての i に対して $w_i = 0$ のときは $\lambda = 0$ に固定してよい) ことがわかる．

次に，2 分探索の終了条件について考える．$f(\lambda)$ は区分線形関数であるので微分不可能な点が存在するが，便宜上 $f'(\lambda)$ を点 λ における $f(\lambda)$ の劣勾配を表す関数として以下のように定義する．点 $(\lambda, f(\lambda))$ が区分線形関数 $f(\lambda)$ のある線分の内部 (つまり区分点 (break point) 以外) にあるときはその傾きを $f'(\lambda)$ の値とし，一方 2 つの線分 a と b の交点上にあるときはそれら 2 つの線分の傾きのうち絶対値が小さい方の傾きを $f'(\lambda)$ の値とする．図 2 に例を示す．この図において，$\lambda = \lambda_{ab}$ であるとき，関数 $f'(\lambda)$ は線分 a と b の傾きのうち小さい方，すなわち線分 a の傾きをとる．2 分探索は $f'(\lambda)$ が正の点 λ_R と負の点 λ_L の 2 点の情報を保持し，その中点 $\lambda_M = (\lambda_R + \lambda_L)/2$ に対して $f'(\lambda_M) > 0$ ならば $\lambda_R := \lambda_M$，$f'(\lambda_M) < 0$ ならば $\lambda_L := \lambda_M$ と更新する，という操作の反復である ($f'(\lambda_M) = 0$ が成立した場合は λ_M が最適な λ の値を与えることが結論できるので 2 分探索をただちに終了できる)．保持している 2 点 λ_R と λ_L のそれぞれにおいて点 λ_R(点λ_L) で関数 f に接する傾き $f'(\lambda_R)$(傾き $f'(\lambda_L)$) の直線を求めることができる．これらの 2 本の直線の交点を (λ_{LBF}, LB) とする (図 2 参照)．LBF は $\min_{\lambda \geq 0} f(\lambda)$ の下界を与える．提案手法では LBF $- f(\lambda)$ がパラメータ δ 以下となったときに 2 分探索を終了する．第 7 節の計算実験では $\delta = 0.1$ とした．

クリーク分割 Q に基づいて上界を計算するアルゴリズムの詳細をアルゴリズム UBCP(Q)(upper bound by clique partition の略) として以下にまとめる．

アルゴリズム UBCP(Q)

Step 1. 全ての i に対して $w_i = 0$ ならば $f(0)$ を出力して終了．さもなければ $\lambda_L := 0$, $\lambda_R := \max\{p_i/w_i \mid w_i > 0\}$ とする．

Step 2. λ_L に対する問題 $DCP(Q)$ の最適解の目的関数値 $f(\lambda_L)$ と劣勾配 $f'(\lambda_L)$ を求め，$f_L := f(\lambda_L)$, $f'_L := f'(\lambda_L)$ とする．

Step 3. λ_R に対する問題 $DCP(Q)$ の最適解の目的関数値 $f(\lambda_R)$ と劣勾配 $f'(\lambda_R)$ を求め，$f_R := f(\lambda_R)$, $f'_R := f'(\lambda_R)$ とする．

Step 4. 点 (λ_L, f_L) を通る傾き f'_L の直線と，点 (λ_R, f_R) を通る傾き f'_R の直線の交点 (λ_{LB}, LBF) を求める．

Step 5. $\lambda := (\lambda_L + \lambda_R)/2$ とする．

Step 6. λ に対する問題 $DCP(Q)$ の最適解の目的関数値 $f(\lambda)$ と劣勾配 $f'(\lambda)$ を求める．

Step 7. もし $(f(\lambda) - \text{LBF}) \leq \delta$ なら，$f(\lambda)$ を出力して終了する．

Step 8. もし $f'(\lambda) = 0$ なら，$f(\lambda)$ を出力して終了する．

Step 9. もし $f'(\lambda) < 0$ なら，$\lambda_L := \lambda$, $f_L := f(\lambda)$, $f'_L := f'(\lambda)$ とする．そうでないなら $\lambda_R := \lambda$, $f_R := f(\lambda)$, $f'_R := f'(\lambda)$ とする．

Step 10. Step 4 へ戻る．

この上界値計算法は，2 分探索の反復回数を α とすると $O(\alpha n)$ 時間で実行することができる．第 7 節の計算実験に用いた問題例に対しては，反復回数 α は平均十数回程度に収まった．

なお，Q がクリーク分割であるときは，制約 (7) は GUB(generalized upper bound) 制約となるが，このような制約のついたナップサック問題は，多重選択ナップサック問題 (multiple–choice knapsack problem) などと呼ばれている ([10],2.12 節)．この問題に対する研究結果より，理論的には問題 $P(Q)$ の線形計画緩和問題の最適値を $O(n \log n)$ 時間で計算できることがただちに言える [8]．しかし，α が小さい場合には，その方法を実装したとしても大きな改善は見込めない．それに加え，そのようなアルゴリズムは UBCP(Q) に比べて複雑であり，実装が容易でないため，今回の実験には用いなかった．

3.2 クリーク分割の生成

本節ではクリーク分割の生成法を説明する．提案アルゴリズムでは，ある頂点のみで構成されるクリークを作り，それを極大化するという動作をクリークに属していない頂点がなくなるまで反復するという方法を採る．その際に，単位重さあたりの価値が近い頂点を同じクリークに含むことで上界の改善が期待できるため，単位重さあたりの価値の降順に上述の動作を行う．アルゴリズムの詳細を以下に MakeCP としてまとめる．

アルゴリズム MakeCP

Step 1. $k := 0$ とする．

Step 2. $k := k + 1$ としたのち $C_k := \emptyset$ する．

Step 3. まだクリーク $C_1, \ldots, C_k$ のいずれにも含まれていない頂点のうち最も頂点番号の小さい頂点をクリーク C_k に加える．

Step 4. もしクリーク C_k に含まれる全ての頂点との間に辺を持ち，まだどのクリークにも所属していない頂点があるなら，その中で頂点番号の最も小さいものをクリーク C_k に加えたのち Step 4 へ戻る．そうでないなら Step 5 へ進む．

Step 5. もし V のどの頂点もクリーク $C_1, \ldots, C_k$ のいずれかに所属している (すなわち $C_1 \cup \cdots \cup C_k = V$) なら，現在のクリーク分割 $\{C_1, \ldots, C_k\}$ を出力して終了する．そうでないなら Step 2 に戻る．

このアルゴリズムは，以下の方法を用いることで $O(n+m)$ 時間で実行することができる．まず，全ての頂点についてカウンタを作成し，この値を 0 とする．クリーク C_k に頂点を追加するたびに，追加した頂点に隣接する頂点のおのおののカウンタに 1 を加算したのち，カウンタの値がクリーク C_k の現在のサイズ $|C_k|$ と同じ頂点のうち，最も頂点番号の小さい頂点を次にクリークに加える候補とする，という操作を反復する．この操作は，各反復ごとに，追加した頂点の隣接リストを 1 回走査することで実現できるので，その頂点の次数のオーダーで可能である．カウンタの値がクリークサイズと一致する頂点がないとき，Step 4 において追加できる頂点が存在しないことが判定できる．このとき，最後に作成したクリーク C_k 内の全ての頂点について，隣接する全ての頂点のカウンタを 0 にする．これを全ての頂点がクリークに属するまで行えば，全頂点の隣接リストを高々2 回走査することで計算が終了するので，計算時間は $O(n+m)$ 時間となる．

4 森状のクリークに対する上界値計算法

本節では森状のクリークに基づく上界値計算法を提案する．クリーク制約による定式化 $P(Q)$ において，クリークの接続関係が森状になっている場合，その線形計画緩和問題の双対問題が特殊な構造を持ち，その最適解を容易に計算できる．本節ではまずこの性質を示したのち，このアイデアに基づく上界値計算法を提案する．

4.1 森状のクリークに対する双対問題

クリークの接続関係を表現するため，クリーク集合 Q に対応してグラフ $G(Q)$ を以下のように定義する．まず，Q に含まれるクリークのおのおのに対してグラフ $G(Q)$ の頂点を対応させる．そして，Q 内の 2 つのクリークが共通のアイテムを含むときかつそのときに限りそれらのクリークに対応する頂点間に辺があるものとする．すなわち，集合 $E(Q)$ を

$$E(Q) = \{(C_k, C_l) \mid C_k, C_l \in Q, k \neq l,\ C_k \cap C_l \neq \emptyset\}$$

と定義すると，$G(Q) = (Q, E(Q))$ である．森状のクリークに対する上界値計算法では，

- V のどの頂点も Q 内の 1 つもしくは 2 つのクリークに属する

- グラフ $G(Q)$ が閉路を持たない

という2つの条件を満たすようにクリークの集合 Q を生成し，双対問題を解くことで上界値を計算する．問題 $P(Q)$ に対する双対問題は以下のように定式化することができる:

$D(Q)$

$$\text{minimize} \quad g(\lambda, y) := c\lambda + \sum_{k=1}^{r} y_k \tag{13}$$

$$\text{subject to} \quad w_i\lambda + \sum_{k:i\in C_k} y_k \geq p_i,\ \forall i \in V, \tag{14}$$

$$\lambda \geq 0,\ y_k \geq 0,\ \forall k : C_k \in Q. \tag{15}$$

この問題において，λ をある値 $\hat{\lambda}$ に固定すると，$\sum_k y_k$ を最小化することで問題 $D(Q)$ の $\lambda = \hat{\lambda}$ の場合の最適解を $O(n)$ 時間で得ることができる．具体的には，クリークの接続関係は森の構造を持つので，葉から順に y_k をとり得る値の最小値に決定していくことで $\sum_k y_k$ を最小化することができる (この詳細については後述する)．その際，3節と同様にこの y_k の値を $\hat{\lambda}$ の関数ととらえ，$y_k(\hat{\lambda})$ と記すことにする．このように考えると問題 $D(Q)$ の目的関数 $g(\lambda, y)$ を $c\lambda + \sum_{k=1}^{r} y_k(\lambda)$ のように λ の関数として考えることができる．便宜上これを $g(\lambda)$ と記す．以下では，$\lambda = \hat{\lambda}$ と固定したときの $y_k(\hat{\lambda})$ の計算方法の詳細を説明する．上述の2つの条件を満たすようにクリークを生成すると，クリーク集合 Q はグラフ $G(Q)$ 上で森状となり，その各連結成分は木状となる．グラフ $G(Q)$ における各木に対して，適当に根となる頂点を決め，根付き木にすることで親子関係を定義する．まず，あるひとつの頂点 C_k が葉である場合には，$y_k(\hat{\lambda}) = \max\{0, \max\{p_i - w_i\hat{\lambda} \mid i \in C_k \setminus (\cup_{l\neq k} C_l)\}\}$ とする．一方，C_k が1つ以上子を持つ場合，C_k の子 $C_{k_1}, C_{k_2}, \ldots, C_{k_s}$ の添え字集合を $H(k) = \{k_1, \ldots, k_s\}$ とすると，$l = k_1, \ldots, k_s$ のおのおのに対して $y_l(\hat{\lambda})$ が決定済みならば，$y_k(\hat{\lambda}) = \max\{0,\ \max\{p_i - w_i\hat{\lambda} - y_l(\lambda) \mid i \in C_k \cap C_l, l \in H(k)\},\ \max\{p_i - w_i\lambda \mid i \in C_k \setminus (\cup_{l\neq k} C_l)\}\}$ とする．たとえば深さ優先探索を利用するなどの方法により，葉から順に上述の計算を行うことで根付き木に属する全ての頂点に対し $y_k(\hat{\lambda})$ を決定することができる．この決定方法を全ての木について繰り返す．

次にこの決定法の最適性を考える．第3節で記述したように，λ をある値 $\hat{\lambda}$ に固定したとき，ひとつのクリーク C_k にしか属さないアイテム i はそのような k に関して y_k に対する下界制約を与える ((4.2)式において変数が y_k ひとつしか現れないため)．このようなアイテムによる下界制約のみを満たした上で各 y_k がとりうる最小値 $\max\{0, \max\{p_i - w_i\hat{\lambda} \mid i \in C_k \setminus (\cup_{l\neq k} C_l)\}\}$ を $L_k(\hat{\lambda})$ とおき，この y_k に対する下界 $L_k(\hat{\lambda})$ からの y_k の増分をあらためて変数 $Y_k = y_k - L_k(\hat{\lambda})$ と定義する．また，簡単のため，以下のような $\tilde{p}_i(\hat{\lambda})$ を各アイテム i に対し定義する．アイテム i が2つのクリーク C_k と C_l に含まれるならば $\tilde{p}_i(\hat{\lambda}) = p_i - w_i\hat{\lambda} - L_k(\hat{\lambda}) - L_l(\hat{\lambda})$ とし，ちょうど1つのクリークに含まれるならば $\tilde{p}_i(\hat{\lambda}) = 0$ とする．y_k から Y_k への変数変換により，λ をある値 $\hat{\lambda}$ に固定したときの $\sum_k y_k(\hat{\lambda})$ の最小値を以下の問題 $DCFY(Q)$ を解くことで得られた最適な Y_k に対して $\sum_k y_k = \sum_k (Y_k + L_k(\hat{\lambda}))$ とすることで得ることができる．

$DCFY(Q, \hat{\lambda})$

$$\text{minimize} \quad \sum_{k=1}^{r} Y_k \tag{16}$$

$$\text{subject to} \quad Y_k + Y_l \geq \tilde{p}_i(\hat{\lambda}),$$

$$\forall i,\ k,\ l : i \in C_k,\ i \in C_l,\ k \neq l \tag{17}$$

$$Y_k \geq 0,\ \forall k : C_k \in Q. \tag{18}$$

上述の方法に従って y_k を決定していく過程を Y_k に変換して考えると，葉となるクリーク C_k に対しては $Y_k = 0$ となる．葉ではないクリーク C_k に対しては，$Y_k = \max\{0, \max\{\tilde{p}_i(\hat{\lambda}) - Y_l \mid i \in C_k \cap C_l,\ l \in H(k)\}\}$ となる．このように各 Y_k を定めた解が問題 $DCFY(Q, \hat{\lambda})$ の実行可能解であることは明らかである．Y_k の決定において有効制約となったものに対応する i (すなわち max を実現した i であり，これを以下ではタイトな i と呼ぶ) を考える．ただし，1つの k に対してそのような i が2つ以上存在する場合は，その中の1つを任意に選ぶものとする．グラフ $G(Q)$ において，そのようなタイトな i に対応する辺を太線で表示した例を図3に示す．このとき，グラフ $G(Q)$ には Y_k の決定において有効制約となった i に対応する太線の辺により，内点から葉へのパスがいくかでき，それらは互いに頂点を共有しない．このようなパス上の親子関係をもつ任意の2頂点は，親である頂点 C_k と子である C_l とその2頂点を結ぶ辺に対応するタイトな i について，$Y_k + Y_l = \tilde{p}_i(\hat{\lambda})$ が成り立つ．あるクリーク C_k から子へ向かう辺の中にタイトな i を含むものが存在

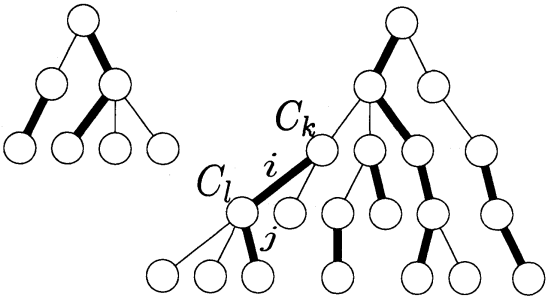

図 3. グラフ $G(Q)$

しない (つまり C_k から下には太線の辺がない) 場合，C_k をパスの下端といい，親へ向かう辺の中にタイトな i がない (つまり C_k から上に向かう辺は太線の辺でない) 場合，C_k をパスの上端ということにする．上にも下にも太線の辺が接続していないものは下端でも上端でもある．ある C_k が下端であれば，$Y_k = 0$ である．次に，問題 $DCFY(Q, \hat{\lambda})$ の双対問題を考えると以下のように定式化することができる．

$PCFY(Q, \hat{\lambda})$

$$\text{maximize} \quad \sum_{i=1}^{n} \tilde{p}_i(\hat{\lambda}) x_i \tag{19}$$

$$\text{subject to} \quad \sum_{i \in C_k} x_i \leq 1, \ \forall C_k \in Q, \tag{20}$$

$$0 \leq x_i \leq 1, \ i = 1, \ldots, n, \tag{21}$$

$$x_i = 0, \ \forall i : |\{k \mid i \in C_k\}| \leq 1. \tag{22}$$

問題 $PCFY(Q, \hat{\lambda})$ において，解 $x = (x_1, \ldots, x_n)$ を以下のように決定する．タイトな i については，上述したパスにおいて，根に近い辺から順に x_i の値を 1,0,1,0 と交互に決定する．一方タイトでない i (グラフ $G(Q)$ のどの辺にも含まれない (つまり 1 つのクリークにしか含まれない) ような i や，パスに含まれない i) については $x_i = 0$ とする．たとえば，図 3 においてタイトな i と j をそれぞれ含む 2 辺より成るパスに対しては，$x_i = 1, x_j = 0$ とする．このように決定することで，パスに含まれる下端以外の任意の頂点には，丁度 1 つ $x_i = 1$ となる i を含む辺が接続する．下端となるクリーク C_k に対しては $x_i = 1$ となる i を含む辺が接続するか否かはパス長に依存するが，前述の通り常に $Y_k = 0$ である．i を含む太線の辺で C_k と C_l が結ばれているとき，$Y_k + Y_l = \tilde{p}_i(\hat{\lambda})$ が成立することは上述の通りであるが，$x_i = 1$ となる i を含む辺（つまり太線の辺）全てに関してこの式の両辺の和をとると，左辺は下端以外の全クリークに対する変数 Y_k の和を取ったものに等しい (下端に対応する Y_k が和に参加することもありうるがその寄与は常に 0 である) ため，結局 $\sum_{k=1}^{r} Y_k = \sum_{i=1}^{n} x_i \tilde{p}_i(\hat{\lambda})$ となり，問題 $DCFY(Q, \hat{\lambda})$ に対する実行可能解と問題 $PCFY(Q, \hat{\lambda})$ に対する解の目的関数値が一致する．また，グラフ $G(Q)$ において各頂点の周りの辺には，$x_i = 1$ となるような i が多くとも 1 つしか存在せず，グラフ $G(Q)$ のどの辺にも含まれないような i については $x_i = 0$ であるので，この方法で求めた解 x が $PCFY(Q, \hat{\lambda})$ 実行可能解であることがわかる．以上のことから，上述の方法で決定した各 Y_k が最適であることわかる．これに伴い，各 y_k も最適であるといえる．

次に，$g(\lambda)$ が下に凸であることを示す．まず，一般的な線形計画問題において 1 つの変数 λ のみをとり出して表記した下記のような問題 DLP を考える．

$$DLP \quad \text{minimize} \quad cx + d\lambda \tag{23}$$

$$\text{subject to} \quad Ax + e\lambda \geq b. \tag{24}$$

(x は変数のベクトル，λ は 1 つの変数，c, e, b は適当なサイズの定数ベクトル，d はスカラー定数，A は適当なサイズの定数行列である.) 問題 DLP において，$\lambda = \lambda_1$ と固定したときの最適値を $h(\lambda_1)$，最適解を x_1 と定義する．同様に $f(\lambda_2)$，x_2 を定義する．

$$x_3 = \frac{x_1 + x_2}{2}, \ \lambda_3 = \frac{\lambda_1 + \lambda_2}{2} \tag{25}$$

を考えると，

$$\begin{aligned} Ax_3 + e\lambda_3 &= A\left(\frac{x_1 + x_2}{2}\right) + e\left(\frac{\lambda_1 + \lambda_2}{2}\right) \\ &= \frac{1}{2}\{(Ax_1 + e\lambda_1) + (Ax_2 + e\lambda_2)\} \\ &\geq \frac{1}{2}(b + b) = b \end{aligned} \tag{26}$$

より，(x_3, λ_3) は問題 DLP の実行可能解であることがわかる．その目的関数値は

$$\begin{aligned} cx_3 + d\lambda_3 &= c\left(\frac{x_1 + x_2}{2}\right) + d\left(\frac{\lambda_1 + \lambda_2}{2}\right) \\ &= \frac{1}{2}\{(cx_1 + d\lambda_1) + (cx_2 + d\lambda_2)\} \\ &= \frac{1}{2}\{h(\lambda_1) + h(\lambda_2)\} \end{aligned} \tag{27}$$

となる．$h(\lambda_3)$ を $\lambda = \lambda_3$ と固定したときの最適値であるとすると，(x_3, λ_3) は実行可能解であるので，

$$cx_3 + d\lambda_3 \geq h(\lambda_3) \tag{28}$$

となる．よって，(4.15) と (4.16) より

$$\frac{h(\lambda_1) + h(\lambda_2)}{2} \geq h\left(\frac{\lambda_1 + \lambda_2}{2}\right) \tag{29}$$

が得られるが，λ_1 と λ_2 は任意なので，問題 DLP において λ を固定したときの最適値を表す関数 h は下に凸であることがわかる．これにより $g(\lambda)$ も下に凸であるといえる．このことから，第 3 節と同様に 2 分探索で $g(\lambda)$ の最適値を発見することができる．

λ の探索範囲について考える．$\lambda^* = \max_{i:w_i \neq 0} p_i/w_i$ とすると，これ以上の任意の λ に対して最適な y_k は，

$$\text{minimize} \quad \sum_{k=1}^{r} y_k \tag{30}$$

$$\text{subject to} \quad \sum_{k:i \in C_k} y_k \geq p_i, \ \forall i : w_i = 0 \tag{31}$$

$$y_k \geq 0, \ \forall k : C_k \in Q, \tag{32}$$

の解である．すなわち，$\forall \lambda \geq \lambda^*$ において，問題 $D(Q)$ の最適値は単調非減少であるので，$0 \leq \lambda \leq \lambda^*$ のみを調べれば $g(\lambda)$ の最適値を得ることができる．

以上の事から，アルゴリズム UBCP(Q) と同様のアルゴリズムを用いることで $g(\lambda)$ の最適値を計算することができる．このアルゴリズムを UBCF(Q) と呼ぶ (upper bound by clique forest の略)．

4.2 森状のクリークの生成

本節では森状のクリークの生成法を説明する．まず前節のクリーク分割を求めるアルゴリズム MakeCP を実行し，次にそれらのクリークを閉路ができないように接続するようなクリークを追加していくという方法をとる．また，接続関係をこのように増やすために追加するクリークの数を大きくする方が，追加するクリークの 1 つ 1 つを大きくするよりも制約を強める効果が高いことが予備実験において確認できたため，クリークの接続関係を作るために追加するクリークは全て 2 頂点のみで構成されるものとした．アルゴリズムの詳細は以下の通りである．

アルゴリズム MakeCF

Step 1. アルゴリズム MakeCP を実行する．得られたクリークの数を r として $k := r$ とする．
Step 2. $k := k + 1$ とする．
Step 3. まだ 1 つのクリークにしか含まれていない頂点で，その頂点との間に辺を持つ頂点の中にまだ 1 つのクリークにしか含まれていないような頂点があり，その 2 頂点より成るクリークを Q に追加してもクリークの接続関係が閉路を持たないならば，その 2 頂点より成るクリークを C_k としたのち，Step 2 に戻る．
Step 4. クリーク集合 $Q = \{C_1, \ldots, C_{k-1}\}$ を出力して終了する．

森状のクリーク生成は以下の方法で実行することで $O(n+m)$ 時間で実行することができる．まず，クリーク分割については前節で説明したように $O(n+m)$ 時間で実行することができる．その後のクリークを追加する操作については，まず MakeCP で生成した各クリーク C_k に対しマーク MC_k を用意し，その値を 0 とする．次に，各頂点 i に対しマーク MI_i を用意し，その値を 0 とする (MC_k は MakeCP で生成されたクリークのみに対して用意し，その後追加されたサイズ 2 のクリークに対しては考えない)．頂点番号の小さい頂点から順にすべての頂点 i に対し以下のアルゴリズムを実行する．

アルゴリズム DFSCF(i, Q)

Step 1. もし $MI_i = 1$ ならば終了する．
Step 2. $MI_i := 1$ とする．
Step 3. $MC_k := 1$, $k : i \in C_k$ とする．
Step 4. もし，頂点 i に隣接し，$MC_l = 0$, $l : j \in C_l$ であるような頂点 j があるならば，そのような j のうち最も小さい j に対し，頂点 i, j で構成されるクリークを Q に加え，アルゴリズム DFSCF(j, Q) を実行する．
Step 5. 頂点 i と同じクリークに属する頂点 j 全てに対しアルゴリズム DFSCF(j, Q) を実行する．

アルゴリズム DFSCF(i, Q) は各頂点 i に対し高々隣接する頂点の数 +1 回しか実行されず，2 回目以降

の実行では (すなわち $MI_i = 1$ となったあとで再度呼び出された場合には) 定数時間で終了する．1回目の実行では頂点の隣接リストを高々2回走査するだけで計算できるので，全体の計算時間は $O(n+m)$ 時間である．

5 発見的解法

本節では，排他制約付きナップサック問題に対する発見的解法について記述する．下記の2つのアルゴリズムを実行することにより，排他制約付きナップサック問題の実行可能解を容易に生成することができる．

5.1 貪欲法

貪欲法について説明する．貪欲法は n 回の反復より成り，i 回目の反復ではアイテム i を解に含むか否かを以下のように決定する．アイテム i を解に追加しても解が実行可能であるならばアイテム i を解に含むと決定し，そうでないならばアイテム i を解に含まないと決定する．貪欲法は n 回の反復の後，解 $x = (x_1, \ldots, x_n)$ を出力する．アルゴリズムの詳細は以下の通りである．

アルゴリズム Greedy

Step 1. $x_j := 0, \forall j \in V$ および $i := 0$ とする．
Step 2. $i := i + 1$ とする．もし $i > n$ ならば，解を出力し終了する．
Step 3. もし $\sum_{j=1}^{i-1} w_j x_j + w_i > c$ ならば，$x_i := 0$ として Step 2 へ戻る．
Step 4. もし $\exists j < i, x_j = 1, (i,j) \in E$ ならば，$x_i := 0$ として Step 2 へ戻る．
Step 5. $x_i := 1$ として Step 2 へ戻る．

アルゴリズムの実行において，$x_i := 1$ としたときに，頂点 i に隣接する頂点全てにマークをつけ，$\sum_{j=1}^{i-1} w_j x_j$ の現在の値を保持することで，アルゴリズム Greedy は $O(n+m)$ 時間で実行することができる．

5.2 局所探索法

局所探索法について説明する．局所探索法は，何らかの方法ですでに得られた解を改善する操作である．便宜上以下では「アイテム i が解に含まれる」とは「$x_i = 1$」を表し，「アイテム i が解に含まれない」とは「$x_i = 0$」を表すとする．用いた局所探索法では以下のように操作を行った．解 x に含まれるアイテムをひとつ解 x から削除し，解 x に含まれていなかったアイテムをひとつ加えることによって得られる解の集合を近傍 $N(x)$ とおく．局所探索法は，近傍解 $y \in N(x)$ の中に実行可能でしかも $z(y) > z(x)$ を満たすものが存在するならば $x := y$ とおきかえる操作を繰り返し，近傍内にそのような解が存在しなくなったら終了する．アルゴリズムを以下にまとめる．

アルゴリズム LocalSearch(x)

Step 1. もし $z(y) > z(x)$ であるような実行可能解 $y \in N(x)$ が存在するなら Step 2 へ進む．そうでないなら解 x を出力したのち停止する．
Step 2. $x := y$ として Step 1 へ戻る．

6 分枝限定法

本節では，分枝限定法 (branch-and-bound method) により最適解を求める方法を記述する．本研究の分枝限定法には，第3節と4節で提案した上界値計算法を取り入れている．

分枝限定法は，問題をいくつかの小規模な問題に分割し，その全てを解くことで等価的に元の問題を解くという考え方に基づいている．本研究では，小規模な問題への分割は，ある1つの変数 x_i の値を0または1に固定し，それぞれの場合を個別に考察することによって実現する．このように問題を分割する操作を分枝操作 (branching operation) という．分枝操作を繰り返し行うことで，すべての場合を列挙することができるが，その過程は生成木と呼ばれる根付き木を用いて表現できる．生成木において，根は元の問題に対応し，その2つの子はそれぞれある変数の値を0または1に固定した2つの場合に対応する．その他の頂点も同様である．よって，内部の頂点は根からその頂点への路に対応していくつかの変数の値を0か1に固定した問題に対応し，葉は全ての変数の値が定まった解に対応する．分枝限定法では，実際に全ての葉を列挙するわけではなく，その一部分だけを生成する．生成木のうち実際に生成された頂点のみから成るものを探索木 (search tree) という．ある部分問題

(頂点) に対して，(i) その最適値，(ii) その部分問題が実行不可能であること，(iii) その部分問題の最適解が元の問題の最適解とはならないこと，のいずれかが分かれば，その部分問題をこれ以上調べる必要はなく，その下の節点 (子孫と呼ぶ) の探索を省略できる (終端する (terminate) という)．これを限定操作 (bounding operation) と呼ぶ．本研究では，部分問題を終端するための条件を以下の 2 つの場合とした．

- その部分問題の実行可能解が存在しない場合．
- その部分問題の最適解が元の問題の最適解とはならない場合．

2 つめの条件は，具体的には部分問題の上界が目的関数の暫定値 LB 以下である場合にその条件に当てはまると判断する (一般に上界値テストと呼ばれる)．探索木の探索方法については，本研究では最後に生成した部分問題を優先して探索する深さ優先探索を用いた．なお，最後に生成した 2 つのうち，$x_i = 1$ と固定したほうを先に探索する．本研究での分枝操作の変数の選び方は，まだ固定していない変数のうち添え字の小さいものから順に選んでいく．以上のルールで探索木を生成することでよい暫定解をはやく発見できると期待できる．

分枝限定法は入力として下界 LB と i を与える．引数 i は $x_1, x_2, \ldots, x_{i-1}$ が 0 か 1 に固定されていることを示す．また，このアルゴリズムは，変数を固定しながら再帰的に実行されるアルゴリズムで，探索木を調べ終え，最適解が発見できたときには最適解を出力する．本研究の分枝限定法の詳細を以下に示す．なお，$z(x)$ は問題 P に対する解 x の目的関数値である．

アルゴリズム BB(LB,i)

Step 1. 現在 $x_i = 0$ と固定されているのであれば Step 5 に進む．さもなければ x_i:=1 とする．頂点 i との間に辺をもつ全ての頂点 $j(> i)$ に対し，$x_j := 0$ と固定する．

Step 2. もし $i = n$ であり，$z(x)$ >LB ならば，LB:= $z(x)$ とする．

Step 3. もし $i \neq n$ であり，部分問題の終端条件のどれにも当てはまらないならば，LB と $i+1$ を入力としてアルゴリズム BB(LB, i) を実行する．

Step 4. 頂点 i との間に辺をもつ全ての頂点 $j(> i)$ に対し，Step 1 で 0 に固定した x_j を自由変数に戻す．

Step 5. x_i:=0 とする．

Step 6. もし $i = n$ であり，$z(x)$ >LB ならば，LB= $z(x)$ とする．

Step 7. もし $i \neq n$ であり，部分問題の終端条件のどれにも当てはまらないならば，LB, $i+1$ を入力としてアルゴリズム BB(LB, i) を実行する．

Step 8. もし $i = 1$ であり，最適解が得られたならば，それを出力し終了する．そうでないならば，実行不可能であったことを出力して終了する．

部分問題を終端するための条件の判定において，上界を計算する際に第 3 節および 4 節で提案した方法を用いる場合，分枝限定法の実行の前にアルゴリズム MakeCP もしくは MakeCF を実行してクリーク集合 Q を用意する．本研究では，このようにして最初に作成したクリーク集合 Q を分枝限定法の実行中を通して利用し，途中で取り直すことはしないという方法を採った．(ただし変数の固定によってグラフに不要な頂点ができるなどの変化は適宜計算に反映している．) また，上界値計算に第 3 節あるいは 4 節で提案した方法を用いる場合，UBCP(Q) または UBCF(Q) の実行において LBF>LB である LBF が得られたならば，上界値テストによってその部分問題を終端できないことが結論できるので即座に上界の計算を終了する．

次に分枝限定法をより効率よく行うための手法，インターバルサーチについて説明する．インターバルサーチとは，暫定解よりも良い下界を分枝限定法で用いることにより，分枝の回数を抑える方法であ

る．この方法を用いると，分枝限定法の分枝の数が大幅に減るため計算の高速化を図ることができる．しかし，用いた下界が最適解よりも大きい場合には最適解を含む分枝を限定してしまい解を発見することができない．この場合には用いる下界の値を暫定解に近づけることにより下界の値が最適値を下回った場合に分枝限定法で最適解を発見することができる．この方法と分枝限定法を併用するアルゴリズムの全体の流れを以下にまとめる．

アルゴリズム ISBB

Step 1. Greedy を実行することで初期解 x を生成し，LocalSearch(x) により解 x を改善する．
Step 2. LB:= $z(x)$ とする．
Step 3. クリークを生成する (アルゴリズム MakeCP もしくは MakeCF を実行する)．
Step 4. 提案した計算法により上界を求め，これを UB とし，T :=UB とする．もし LB$\geq T$ ならば，最適解を出力して終了する．
Step 5. $T := (T+\mathrm{LB})/2$ とする．
Step 6. アルゴリズム BB(T, 1) を実行する．
Step 7. 最適解を発見できればそれを出力し終了する．さもなければは Step 5 へ戻る．

7 計算結果

7.1 実験環境と問題例

第 3 節と第 4 節で提案した上界値計算法による上界の精度とその計算に要する時間，および第 6 節の分枝限定法の性能を調べるために計算実験を行った．提案したこれらのアルゴリズムは C 言語で実装した．また，比較対象には汎用の混合整数計画ソルバーである CPLEX 10.0 を用いた．本節の計算実験に用いた計算機は Dell Precision 470 (Xeon 3GHz, 2MB cache, 8GB memory) である．問題例の生成には Yamada ら [14] の方法を用い，文献 [14] の著者から入手したプログラムを用いて問題例を生成した． この方法で生成される問題例には 3 タイプあり，アイテム i の重み w_i と価値 p_i の相関の強さに応じて uncor, weak, strong と呼ばれる．タイプ uncor では，w_i と p_i はいずれも 1 から u (パラメータ) の間の整数から一様にランダムに選ばれる．タイプ weak では，まず w_i を 1 から u の間の整数からランダムに選んだのち，w_i の値に 1 から 10 の整数からランダムに選んだ値を加算したものを p_i とする (これを各 i に対して独立に行う)．タイプ strong では，各 i に対して，w_i を 1 から u の間の整数からランダムに選んだのち，w_i の値に 10 を加算したものを p_i とする．排他制約を表す集合 E の生成には，ランダムグラフの辺集合の生成方法を用いる．具体的には，辺密度を定めるパラメータ μ ($\mu < 1$) を用いて，各対 $(i,j) \in V \times V (i < j)$ に対して独立に試行を行い，確率 μ で (i,j) を E に入れる．また，上界値計算法におけるパラメータ δ については全ての計算において 0.1 とした．

7.2 上界の計算結果

提案した上界値計算法による上界の計算結果を表 1–4 に示す．なお，本節の実験では，問題例の生成に用いるパラメータ u を 1000 とした．表中，CP は第 3 節で提案した方法 (アルゴリズム MakeCP によって生成された Q を用いて UBCP(Q) を実行する方法)，CF は第 4 節で提案した方法 (アルゴリズム MakeCF によって生成された Q を用いて UBCF(Q) を実行する方法)，LP は問題 P の線形計画緩和問題の最適値を汎用ソルバー CPLEX で計算した結果である．time[sec] は上界の計算に要した計算時間 (秒)，UB は得られた上界値である．なお，「T.O.」は 30 分計算を行っても計算が終了しなかったことを，「< 0.01」は計算時間が 0.01 秒未満で正確な値の計測ができなかったことを表す．

まず，辺密度が非常に小さい場合の計算結果を表 1，2，3 に示す．辺密度は $4/(n-1)$ であるので，各問題例に対して辺の数はおよそ $2n$ 程度の数となっている．

表 1，2，3 の計算結果では，計算時間については LP(CPLEX) に比べ非常に良い結果が得た．上界値については，問題 P の線形計画緩和を用いるほうが良い結果を得られた．しかし，表 2 と 3 が示すように，タイプ weak と strong の問題例 (すなわち，重さ w_i と価値 p_i に相関があるような問題例) では，CF と LP の上界値の差は 0.1 から 0.4% 程度であった．以上の結果から，辺密度が低い問題例に対しては，LP ソルバーで上界を計算することが困難であるような非常に大規模な問題例に対しても提案手法は上界を高速に計算でき，重さ w_i と価値 p_i に相関があるような問題例では，問題 P の線形計画緩和と同程度の上界値を高速に計算できることがわかっ

表 1. 辺密度の低い問題例に対する上界の計算結果 (タイプ uncor，容量 $c = 250n$)

		time[sec]			UB		
μ	n	CP	CF	LP	CP	CF	LP
$4/(n-1)$	1000	<0.01	0.01	0.09	328,806	317,119	275,269
$4/(n-1)$	5000	0.03	0.03	1.32	1,636,269	1,581,032	1,365,520
$4/(n-1)$	10,000	0.04	0.06	3.56	3,257,727	3,137,602	2,706,960
$4/(n-1)$	15,000	0.07	0.09	7.61	4,894,769	4,702,547	4,070,890
$4/(n-1)$	20,000	0.09	0.13	13.34	6,527,604	6,285,778	5,457,670
$4/(n-1)$	25,000	0.12	0.17	24.46	8,161,850	7,845,602	6,795,490
$4/(n-1)$	30,000	0.14	0.20	30.55	9,805,778	9,424,374	8,182,700
$4/(n-1)$	35,000	0.16	0.24	26.79	11,438,162	11,001,451	9,519,410
$4/(n-1)$	40,000	0.19	0.28	39.64	13,045,818	12,527,608	10,880,800
$4/(n-1)$	45,000	0.22	0.32	50.93	14,689,136	14,136,388	12,243,800
$4/(n-1)$	50,000	0.25	0.35	59.50	16,286,597	15,650,615	13,565,500
$4/(n-1)$	60,000	0.29	0.45	73.62	19,522,814	18,773,844	16,261,300
$4/(n-1)$	70,000	0.35	0.53	101.49	22,780,679	21,885,749	18,973,300
$4/(n-1)$	80,000	0.41	0.65	161.31	26,043,850	25,003,780	21,715,500
$4/(n-1)$	100,000	0.52	0.89	216.28	32,576,887	31,284,230	27,143,800
$4/(n-1)$	125,000	0.63	1.17	309.6	40,757,272	39,162,587	33,988,900
$4/(n-1)$	150,000	0.82	1.53	464.22	48,988,624	47,079,260	40,840,500
$4/(n-1)$	200,000	1.13	2.27	744.62	65,262,876	62,721,845	54,399,000
$4/(n-1)$	300,000	1.77	3.83	1606.58	97,936,834	94,112,462	81,595,700
$4/(n-1)$	500,000	3.21	6.94	T.O.	163,194,545	156,865,242	—
$4/(n-1)$	1,000,000	7.34	15.24	T.O.	326,012,090	313,222,862	—

た．

次に，様々な辺密度のタイプ weak の問題例に対する CF と LP(CPLEX) の計算結果を表 4 に示す．なお，上界値計算においては，第 4 節で提案した手法 CF の方が第 3 節で提案した CP よりも精度が高く，計算時間もそれほど大きくは変わらないことが予備実験により確認できたので，CF に注目して計算実験を行った．

表に示したどの問題例に対しても LP(CPLEX) よりも CF が高速に上界値を計算することが確認できる．上界値についても，辺密度が 5% 以上の問題例については LP(CPLEX) よりも良い結果が得られた．また，タイプ uncor と strong についても類似する結果が得られた．以上のことから，提案手法は辺密度がある程度高い問題例に対しては，問題 P の線計画緩和を用いる方法より良い性能であり，辺密度の低い問題例に対しても高速に同程度の上界値が計算できることがわかった．

7.3 最適解の計算結果

提案した上界値計算法を分枝限定法に取り入れ，最適解の計算を行った．計算結果を表 5，6 に示す．ISBB-CF は，第 4 節で提案した上界値計算法 (アルゴリズム MakeCF によって生成されたクリーク集合 Q を用いて UBCF(Q) を実行する方法) を第 6 節の分枝限定法 BB で用い，インターバルサーチ ISBB に組み込んだものを表す．また，CPLEX は汎用ソルバー CPLEX10.0 の計算結果を示す．time[sec] は最適解の計算に要した計算時間 (秒) である．なお，「T.O.」は 30 分計算を行っても計算が終了しなかったことを意味する．また，「< 0.1」は計算時間が 0.1 秒未満で正確な値の計測ができなかったことを表す．なお，本節の計算実験に用いた問題例の生成においては，パラメータ u を 500 とした．

表 5 と 6 より，辺密度の低い問題例に対しては，CPLEX の方が高速に最適解の計算が行えることがわかる．しかし，提案した上界値計算法が有効であることが 7.2 節で観測された辺密度の高い問題例に対しては，CPLEX よりも高速に最適解を得ることができた．また，そのような問題例の中には，CPLEX では制限時間の 30 分以内に計算が終了しなかったのに対し，ISBB-CF では数分程度で計算が終了するなど大きな差が見られるものも多数存在した．

なお，文献 [2, 14] の分枝限定法の計算結果は，いずれも辺密度の極めて低い問題例に対してのみ報告されている．そのような問題例では汎用解法の CPLEX のほうが本研究で提案する手法よりもすぐれた結果を示しているため，これらの文献の計算結果との比較は割愛した．

表 2. 辺密度の低い問題例に対する上界の計算結果 (タイプ weak, 容量 $c = 250n$)

		time[sec]			UB		
μ	n	CP	CF	LP	CP	CF	LP
$4/(n-1)$	1000	<0.01	<0.01	0.16	256,417	256,114	255,242
$4/(n-1)$	5000	0.02	0.03	3.76	1,281,108	1,279,814	1,275,590
$4/(n-1)$	10,000	0.04	0.06	15.70	2,562,335	2,559,806	2,551,380
$4/(n-1)$	15,000	0.07	0.09	32.56	3,843,794	3,840,035	3,827,220
$4/(n-1)$	20,000	0.09	0.12	56.93	5,125,119	5,120,364	5,103,540
$4/(n-1)$	25,000	0.11	0.16	100.67	6,406,245	6,399,960	6,379,140
$4/(n-1)$	30,000	0.14	0.19	134.70	7,687,442	7,679,820	7,654,740
$4/(n-1)$	35,000	0.17	0.22	585.29	8,969,203	8,960,632	8,930,600
$4/(n-1)$	40,000	0.18	0.26	789.13	10,249,978	10,239,868	10,206,500
$4/(n-1)$	45,000	0.21	0.30	1466.17	11,531,428	11,520,155	11,481,800
$4/(n-1)$	50,000	0.23	0.33	1566.16	12,812,592	12,800,183	12,758,000
$4/(n-1)$	60,000	0.28	0.42	T.O.	15,375,361	15,360,187	—
$4/(n-1)$	70,000	0.34	0.51	T.O.	17,936,666	17,919,027	—
$4/(n-1)$	80,000	0.40	0.58	T.O.	20,499,311	20,479,400	—
$4/(n-1)$	100,000	0.50	0.84	T.O.	25,624,744	25,599,966	—
$4/(n-1)$	125,000	0.63	1.12	T.O.	32,030,763	31,999,652	—
$4/(n-1)$	150,000	0.78	1.45	T.O.	38,437,815	38,400,203	—
$4/(n-1)$	200,000	1.09	2.09	T.O.	51,250,803	51,201,737	—
$4/(n-1)$	300,000	1.71	3.54	T.O.	76,876,137	76,800,650	—
$4/(n-1)$	500,000	3.15	6.18	T.O.	128,122,554	127,998,478	—

表 3. 辺密度の低い問題例に対する上界の計算結果 (タイプ strong, 容量 $c = 250n$)

		time[sec]			UB		
μ	n	CP	CF	LP	CP	CF	LP
$4/(n-1)$	1000	<0.01	0.01	0.18	255,637	255,561	255,090
$4/(n-1)$	5000	0.02	0.03	2.24	1,278,296	1,277,901	1,275,550
$4/(n-1)$	10,000	0.04	0.06	13.56	2,556,772	2,556,067	2,551,140
$4/(n-1)$	15,000	0.06	0.09	27.00	3,835,020	3,833,926	3,826,760
$4/(n-1)$	20,000	0.08	0.12	115.94	5,113,806	5,112,333	5,102,350
$4/(n-1)$	25,000	0.11	0.15	198.29	6,392,380	6,390,592	6,377,710
$4/(n-1)$	30,000	0.13	0.19	305.45	7,670,325	7,668,142	7,653,280
$4/(n-1)$	35,000	0.15	0.22	T.O.	8,948,648	8,946,103	—
$4/(n-1)$	40,000	0.18	0.26	T.O.	10,227,215	10,224,189	—
$4/(n-1)$	45,000	0.21	0.30	T.O.	11,505,323	11,502,068	—
$4/(n-1)$	50,000	0.24	0.34	T.O.	12,784,250	12,780,477	—
$4/(n-1)$	60,000	0.28	0.41	T.O.	15,340,740	15,336,366	—
$4/(n-1)$	70,000	0.32	0.52	T.O.	17,897,576	17,892,407	—
$4/(n-1)$	80,000	0.39	0.62	T.O.	20,453,933	20,447,946	—
$4/(n-1)$	100,000	0.50	0.83	T.O.	25,568,235	25,560,757	—
$4/(n-1)$	125,000	0.62	1.13	T.O.	31,960,910	31,951,848	—
$4/(n-1)$	150,000	0.79	1.45	T.O.	38,352,645	38,341,750	—
$4/(n-1)$	200,000	1.07	2.11	T.O.	51,136,148	51,121,434	—
$4/(n-1)$	300,000	1.71	3.57	T.O.	76,703,555	76,681,575	—
$4/(n-1)$	500,000	3.15	6.37	T.O.	127,841,872	127,805,155	—

表 4. 様々な辺密度の問題例に対する上界の計算結果 (タイプ weak, $c = 250n$)

		time[sec]		UB	
μ	n	CF	LP	CF	LP
0.001	1000	<0.01	0.02	256,787	256,745
0.001	2000	0.01	0.08	512,983	512,522
0.001	3000	0.01	0.27	768,565	767,145
0.001	4000	0.02	1.89	1,023,951	1,020,750
0.001	5000	0.03	4.89	1,278,746	1,274,510
0.005	1000	0.01	0.21	255,557	254,673
0.005	2000	0.01	1.76	510,502	509,262
0.005	3000	0.03	3.24	765,752	764,103
0.005	4000	0.05	3.08	1,020,742	1,018,880
0.005	5000	0.08	4.57	1,275,627	1,273,590
0.01	1000	0.01	0.45	255,204	254,562
0.01	2000	0.02	1.69	510,025	509,255
0.01	3000	0.06	1.89	765,082	764,104
0.01	4000	0.09	4.06	1,019,865	1,018,880
0.01	5000	0.14	8.44	1,274,461	1,273,590
0.05	1000	0.03	0.86	254,148	254,547
0.05	2000	0.11	2.66	492,061	509,255
0.05	3000	0.25	8.41	720,758	764,103
0.05	4000	0.42	18.47	915,857	1,018,880
0.05	5000	0.67	44.44	1,132,194	1,273,590
0.1	1000	0.05	1.05	230,824	254,547
0.1	2000	0.21	6.09	429,569	509,255
0.1	3000	0.47	15.30	613,698	764,103
0.1	4000	0.82	31.58	799,493	1,018,880
0.1	5000	1.31	122.47	966,512	1,273,590
0.5	1000	0.22	4.01	109,868	254,547
0.5	2000	0.98	23.11	196,133	509,255

表 5. 最適解の計算に要した計算時間 (タイプ uncor, $c = 1000,\ 0.001 \leq \mu \leq 0.9$)

		time[sec]	
μ	n	ISBB-CF	CPLEX
0.001	1000	0.6	<0.1
0.002	1000	2.0	<0.1
0.003	1000	2.1	0.1
0.004	1000	3.8	0.1
0.005	1000	0.5	<0.1
0.006	1000	0.6	0.1
0.007	1000	0.4	0.1
0.008	1000	1.5	0.2
0.009	1000	3.4	0.4
0.01	1000	1.0	0.2
0.02	1000	2.0	3.0
0.03	1000	1.9	2.8
0.04	1000	1.3	3.2
0.05	1000	1.5	2.7
0.06	1000	3.6	6.6
0.07	1000	19.6	14.4
0.08	1000	8.7	17.3
0.09	1000	73.9	12.9
0.1	1000	13.7	36.7
0.2	1000	911.0	471.7
0.3	1000	879.7	T.O.
0.4	1000	466.9	T.O.
0.5	1000	338.0	T.O.
0.6	1000	173.0	T.O.
0.7	1000	160.3	T.O.
0.8	1000	69.1	T.O.
0.9	1000	40.0	T.O.

表 6. 最適解の計算に要した計算時間 (タイプ weak, $c = 1000,\ 0.001 \leq \mu \leq 0.9$)

		time[sec]	
μ	n	ISBB-CF	CPLEX
0.001	1000	0.8	<0.1
0.002	1000	1.5	<0.1
0.003	1000	1.6	<0.1
0.004	1000	0.4	<0.1
0.005	1000	8.8	0.1
0.006	1000	2.0	0.1
0.007	1000	2.9	0.2
0.008	1000	5.7	0.2
0.009	1000	3.2	0.1
0.01	1000	7.2	0.2
0.02	1000	8.2	2.8
0.03	1000	8.3	3.5
0.04	1000	13.6	4.9
0.05	1000	20.2	5.1
0.06	1000	51.4	11.6
0.07	1000	87.4	21.0
0.08	1000	110.3	24.4
0.09	1000	71.8	15.5
0.1	1000	893.2	67.0
0.2	1000	744.5	590.23
0.3	1000	519.1	T.O.
0.4	1000	512.7	T.O.
0.5	1000	628.8	T.O.
0.6	1000	205.8	T.O.
0.7	1000	169.1	T.O.
0.8	1000	102.6	T.O.
0.9	1000	58.1	T.O.

8 結論

排他制約付きナップサック問題に対して線形計画緩和問題に対する汎用解法 (LP ソルバー) を必要としない上界の計算法を提案した．計算実験の結果，提案手法は LP ソルバーを用いるよりも高速に計算が行えることが分かった．これにより，既存の方法では上界値を計算することが困難であった非常に大規模な問題例についても計算が可能となった．上界値は，ヒューリスティクスやメタ戦略アルゴリズムなど，最適解を得る保証のないアルゴリズムによって得られた解の精度の実用的な評価にも利用できるので，LP ソルバーでは上界値の計算が困難であるような大規模な問題例に対して精度の高い上界値が得られるようになったことは大いに意義があるといえる．辺密度の高い問題例に対しては，上界値，計算時間ともに，LP ソルバーを用いるよりも優れていることが分かった．また，提案した上界値計算法を分枝限定法に取り入れることで，辺密度の高い問題例に対して，汎用ソルバーとして定評のある CPLEX よりも高速な計算が可能であることを示した．

謝辞

本研究の遂行にあたり，問題例作成プログラムを提供して下さった山田武夫教授，有益なコメントを頂いきました平田富夫教授，久野誉人教授に深くお礼申し上げます．

参考文献

[1] A. Billionnet, A. Faye and E. Soutif, "A new upper bound for the 0-1 quadratic knapsack problem," European Journal of Operational Research, vol.112, pp.664–672, 1999

[2] M. Hifi and M. Michrafy, "Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem," Computers & Operation Research, vol.34, pp.2657–2673, 2005

[3] M. Hifi and M. Michrafy, "A reactive localsearch-based algorithm for the disjunctively constrained knapsack problem," Journal of the Operational Research Society, vol.57, pp.718–726, 2006

[4] T. Ibaraki, Enumerative Approaches to Combinatorial Optimization - Part I and II, Annals of Operations Research, Vols. 10-11, J.C. Baltzer, Basel, Switzerland, 1987

[5] H. Kellerer, U. Pferschy and D. Pisinger, Knapsack Problems, Springer, Berlin, 2004

[6] P. J. Kolesar, "A branch and bound algorithm for the knapsack problem," Management Science, vol.13, pp.723–735, 1967

[7] K. Makino and T. Uno, "New algorithms for enumerating all maximal cliques," Lecture Notes in Computer Science 3111 (SWAT2004, Scandinavia Workshop on Algorithm Theory), Springer–Verlag, pp.260–272, 2004

[8] P. Sinha and A. A. Zoltners, "The multiple–choice knapsack problem," Operations Research, vol.27, pp.503–515, 1979

[9] S. Martello and P. Toth, "An upper bound for the zero-one knapsack problem and a branch and bound algorithm," European Journal of Operational Research, vol.1, pp.169–175, 1977

[10] S. Martello and P. Toth, Knapsack Problems, John Wiley & Sons, Chichester, 1990

[11] H. M. Salkin and C. A. De Kluyver, "The knapsack problem: a survey," Naval Research Logistics Quarterly, vol.22, pp.127–144, 1975

[12] W. Shih, "A branch and bound method for the multiconstraint zero-one knapsack problem," Journal of the Operational Research Society, vol.30, pp.369–378, 1979

[13] L. A. Wolsey, "Valid inequalities for 0-1 knapsacks and MIPs with generalised upper bound constraints," Discrete Applied Mathematics, vol.29, pp.251–261, 1990

[14] T. Yamada, S. Kataoka and K. Watanabe, "Heuristic and exact algorithms for disjunctively constrained knapsack problem," IPSJ Journal, vol.43, pp.2864–2870, 2002