社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－AL－116 (7)
2008／1／23

多点対カット問題に対する集合被覆アプローチに基づく近似解法

木本 大介　柳浦 睦憲　小野 孝男　平田 富夫
名古屋大学

概要. 多点対カット問題はターミナル対が3つ以上の場合，NP 困難であることが知られている．各ターミナル対間の経路を列挙し，その経路をカットするために必要な辺を考えることで，多点対カット問題を集合被覆問題に帰着できるが，本研究では，この考え方に基づいて近似解を求めるアルゴリズムを提案し，その性能を実験的に評価する．ターミナル対間の全経路を列挙するのは現実的ではないので，提案手法では，一部の経路からなる集合被覆問題を考え，そのラグランジュ緩和から得られる情報を利用したヒューリスティクスで近似解を求める．計算実験の結果，提案手法の有効性を確認できた．

A Heuristic Algorithm based on the Set Covering Approach for the Multicut Problem

Daisuke Kimoto　Mutsunori Yagiura　Takao Ono　Tomio Hirata
Nagoya University

Abstract. The multicut problem is known to be NP-hard when the number of terminal pairs is three or more. Multicut problem can be reduced to the set covering problem (SCP) by enumerating paths between each terminal pairs and considering necessary edges to cut such paths. In this paper, we propose a heuristic algorithm based on this idea, in which a Lagrangian heuristic algorithm is used to obtain approximate solutions of SCP. Note that the algorithm maintains a manageable number of paths, because it is impractical to enumerate all paths between each terminal pairs. Through computational experiments, we confirm the effectiveness of our algorithm.

1 はじめに

辺に正のコストを伴う無向グラフと，ターミナル対と呼ばれる頂点対の集合が与えられたとき，各ターミナル対がそれぞれ異なる連結成分に属するようにカットする辺集合のことを多点対カットと呼び，そのようなカットの中で総コスト最小のものを求める問題を多点対カット問題 (multicut problem) と呼ぶ．この問題はターミナル対の数 $k = 1$ のときは最小 s-t カットとなる．また $k = 2$ のときは最大フローを用いたアルゴリズムにより多項式時間で解くことができる [7]．これに対し，$k \geq 3$ においては，NP 困難であることが知られており [2]，$O(\log k)$ 近似のアルゴリズムが存在する [4]．また，各ターミナルを他の全てのターミナルと異なる連結成分に属するようにカットする特殊な場合には，この問題は多分割カット問題と呼ばれ，ターミナルの総数にかかわらず 3/2 近似のアルゴリズムが存在する [6]．

本研究では，各ターミナル対間の経路を列挙し，その経路をカットするために必要な辺を考えることで，多点対カット問題を集合被覆問題に帰着して近似解を求めるアルゴリズムを提案し，その性能を実験的に評価する．ここで集合被覆問題とは，ある集合 P の部分集合の族 $S_1, \ldots, S_n \subseteq P$ とそれらのコストが与えられたとき，部分集合からいくつか選んで P を被覆する総コスト最小の組合せを求める問題である．

多点対カット問題における各ターミナル対間の全経路を集合 P とおき，各枝 e_j が切断する経路すべての集合を $S_j \subseteq P$ とおくことで，多点対カット問題を集合被覆問題に帰着できる．しかし，そのような

経路集合 P のサイズは元の問題規模に対して指数的に大きくなり得るため，その全体を明示的に扱うのは現実的でない．そこで，提案手法は，まずグラフからターミナル対間の経路をいくつか選び，それらのみから定義される集合被覆問題に対する実行可能解を求める．そのような解に対応する辺集合は，選んだ経路はすべて切断するが，それ以外の経路の中には切断されずに残っているものも存在し得る．そこでそのような経路を切断する辺をカットする辺として追加することで，多点対カット問題の実行可能解を得る．また，そのような経路を追加した新たな集合被覆問題を生成し，それを解くことによって多分割カット問題の実行可能解が得られる可能性を高める．これらの操作を繰り返すことでより良い近似解を求める．集合被覆問題もまた NP 困難であるので，解くために，ラグランジュ緩和から得られる情報を利用したヒューリスティクスを用いる．

計算実験の結果，数理計画ソフトである CPLEX よりも平均的に良い結果を得ることができた．

2 多点対カット問題

入力として各辺 $e \in E$ にコスト $c_e > 0$ が付随している無向グラフ $G = (V, E)$ と，ターミナルと呼ばれる指定頂点対の集合 $T = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ が与えられる．このとき，削除すると各ターミナル対が異なる連結成分に属するような辺集合を多点対カットと呼び，総コスト最小の多点対カットを出力する問題を多点対カット問題と呼ぶ．

ターミナル対 (s_r, t_r) 間を結ぶ全ての経路の集合を P_r とし，それら全体の集合を $P = \bigcup_{r=1}^{k} P_r$ とする．また，辺 e と経路 p に対し e が p に含まれていることを $e \in p$ と表す．これを用いると，多点対カット問題は次の整数線形問題として定式化することができる．

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \text{s.t.} \quad & \sum_{e \in p} x_e \geq 1 \quad (p \in P) \\ & x_e \in \{0, 1\} \quad (e \in E) \end{aligned} \tag{1}$$

x_e は辺 $e \in E$ が多点対カットに選ばれるとき 1，それ以外は 0 である．

3 提案アルゴリズム

3.1 提案アルゴリズムの概要

提案アルゴリズムは，ターミナル対間の経路と，それに含まれている辺の関係から多点対カット問題を集合被覆問題に帰着し，近似解を求める．全経路を用いて集合被覆問題を構成するのは，経路数が膨大なため現実的でない．そこで本アルゴリズムでは，一部の経路のみを用いて集合被覆問題を構成するが，その際，辺数の少ない経路を優先的に選ぶことにする．これは，辺数の多い経路は辺数の少ない経路の辺を含んでいる可能性が高いためである．実際，少数の経路を全て切断する辺の集合が全経路を切断する辺の集合となること，つまり多点対カット問題の解になっていることは頻繁に生じる．以下では，探索のために選択した経路の集合を $\hat{P} \subseteq P$ とする．

提案アルゴリズムの概要は次の通りである．まず与えられたグラフ G からターミナル対間の経路を選択し，$\hat{P}$ とする (3.3.1 節)．現在の経路集合 $\hat{P}$ で定義される集合被覆問題を解き，その近似解を得る (3.2 節)．この解が多点対カットになっていなければ，$P \setminus \hat{P}$ の経路を切断する辺を追加していくことで，多点対カットを生成する (3.4 節)．探索の終了条件がみたされなければ経路を $\hat{P}$ に追加 (3.3.2 節) した後，再び集合被覆問題を解き，同様の手続きを反復する．

以下，個々の処理の詳細を示す．

3.2 集合被覆問題

3.2.1 集合被覆問題への帰着

便宜上，E の辺と $\hat{P}$ の経路に番号をつけ，それぞれを e_j $(j = 1, 2, \ldots, n)$, p_i $(i = 1, 2, \ldots, m)$ と記す．変数 x_j $(j = 1, 2, \ldots, n)$ を辺 e_j をカットするとき値 1，それ以外は値 0 をとる 0-1 変数とする．また，a_{ij} を辺 e_j が経路 p_i $(i = 1, 2, \ldots, m)$ に含まれるとき 1，それ以外は 0 と定義する．簡単のため e_j のコスト c_{e_j} を，以下では c_j と記す．$\hat{P}$ の全ての経路をカットする辺集合を見つける問題は，集合被覆

問題として以下のように定式化することができる．

$$\mathrm{SCP}(\hat{P}): \quad \min \quad \sum_{j=1}^{n} c_j x_j$$
$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad (i = 1, \ldots, m)$$
$$x \in \{0,1\}^n$$

$\hat{P} = P$ のとき，この問題は元の多点対カット問題と等価である．集合被覆問題もまた NP 困難であるため，3.2.2 節で説明するラグランジュ緩和を用いて近似解を求める．

3.2.2 ラグランジュ緩和

ラグランジュ緩和とは，条件を緩和して整数計画問題の下界を求めるなどの目的に広く利用される手法である．集合被覆問題 $\mathrm{SCP}(\hat{P})$ のラグランジュ緩和問題を導くため，経路 p_i に対応するラグランジュ乗数 $\lambda_i (\geq 0)$ を導入し，そのベクトルを $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$ とする．集合被覆問題のラグランジュ緩和問題は

$$\min_{x \in \{0,1\}^n} \quad \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} \lambda_i \left(1 - \sum_{j=1}^{n} a_{ij} x_j\right) \quad (2)$$

と定義できる．この問題は，

$$\min_{x \in \{0,1\}^n} \quad \sum_{j=1}^{n} \left(c_j - \sum_{i=1}^{m} \lambda_i a_{ij}\right) x_j + \sum_{i=1}^{m} \lambda_i \quad (3)$$

と表すことができる．ここで，

$$c_j(\lambda) = c_j - \sum_{i=1}^{m} \lambda_i a_{ij} \quad (j = 1, 2, \ldots, n) \quad (4)$$

と置くと，式 (2) は

$$\min_{x \in \{0,1\}^n} \quad \sum_{j=1}^{n} c_j(\lambda) x_j + \sum_{i=1}^{m} \lambda_i \quad (5)$$

と書くことができる．式 (5) の最適解 $x_j(\lambda)$ は，

$$x_j(\lambda) = \begin{cases} 1 & (c_j(\lambda) \leq 0) \\ 0 & (\text{その他}) \end{cases} \quad (6)$$

であり，これを用いると，ラグランジュ乗数ベクトル λ に対する問題 (2) の最適解 $z_{\mathrm{LB}}(\lambda)$ は

$$z_{\mathrm{LB}}(\lambda) = \sum_{j=1}^{n} c_j(\lambda) x_j(\lambda) + \sum_{i=1}^{m} \lambda_i \quad (7)$$

と表すことができる．このように，与えられた λ に対するラグランジュ緩和問題の最適解は機械的に決定することができる．任意の λ $(\lambda_i \geq 0, \forall i)$ に対し，$z_{\mathrm{LB}}(\lambda)$ は $\mathrm{SCP}(\hat{P})$ の最適値の下界を与える．また，$z_{\mathrm{LB}}(\lambda)$ は，元の多点対カット問題の最適解の下界にもなっている．

3.2.3 ラグランジアンヒューリスティック

ラグラジアンヒューリスティックとはラグランジュ緩和をもとに実行可能解 (上界) を求める手法のことである．提案アルゴリズムでは，Caprara ら [1] が提案したヒューリスティクスの基本部分を利用して実行可能解を求める．その手法を以下に述べる．

X を集合被覆として選ぶ辺の集合 (すなわち $e_j \in X \iff x_j = 1$) とする．$\hat{P} = \{p_1, p_2, \ldots, p_m\}$，$E = \{e_1, e_2, \ldots, e_n\}$ と，二値行列 $A = (a_{ij}) \in \{0,1\}^{m \times n}$ に対し

$$I_j = \{p_i \in \hat{P} : a_{ij} = 1\} \quad (j = 1, 2, \ldots, n)$$
$$P^* = \left\{ p_i \in \hat{P} : \sum_{j : e_j \in X} a_{ij} = 0 \right\} \quad (8)$$

とする．相対的にコストの小さい辺を選ぶ基準として，

$$\gamma_j = c_j - \sum_{p_i \in I_j \cap P^*} \lambda_i \quad (9)$$

によって

$$\sigma_j = \begin{cases} \dfrac{\gamma_j}{|I_j \cap P^*|} & (\gamma_j \geq 0) \\ \gamma_j |I_j \cap P^*| & (\text{その他}) \end{cases} \quad (10)$$

で与えられる値を用いる．アルゴリズムを以下にまとめる．

1. $P^* := \hat{P}$，$X := \emptyset$ とする．
2. 各 $e_j \in E \setminus X$ に対して σ_j を計算する．σ_j が最小となる添え字を j^* とする．
3. $X := X \cup \{e_{j^*}\}$，$P^* := P^* \setminus I_{j^*}$ と更新する．
4. $P^* \neq \emptyset$ ならばステップ 2 へ戻る．
5. X とそのコスト $\sum_{j : e_j \in X} c_j$ を出力する．

ある λ に対して上述のアルゴリズムを適用することによって得られた解 X を，以下 $X(\lambda)$ と記す．

3.2.4 劣勾配法

劣勾配法 [3, 5] は，ラグランジュ乗数を，適当な初期値から始め，劣勾配に基づいて逐次更新していく方法であり，ラグランジュ緩和問題から得られる下界値の最大化を目標とする．以下では，問題 SCP($\hat{P}$) に対して探索中に得られた最良の上界を z_{UB} と記す．劣勾配法の反復を開始する時点で必要となる上界 z_{UB} とラグランジュ乗数 λ の初期値についてはそれぞれ 3.2.5 節と 3.2.6 節で説明する．劣勾配法の一般的な手続きは以下の通りである．

現在のラグランジュ乗数を $\lambda^{(h)}$ とし，更新後のラグランジュ乗数を $\lambda^{(h+1)}$ とする．π を $0 < \pi \leq 2$ を満たすパラメータとする．現在のラグランジュ緩和の解に対する劣勾配

$$G_i = 1 - \sum_{j=1}^{n} a_{ij} x_j(\lambda^{(h)}) \quad (i = 1, \ldots, m) \tag{11}$$

を計算する．ステップ長 q を

$$q = \frac{\pi \left(z_{\mathrm{UB}} - z_{\mathrm{LB}}(\lambda^{(h)}) \right)}{\sum_{i=1}^{m} G_i^2} \tag{12}$$

と置く．そして

$$\lambda_i^{(h+1)} = \max\{0, \lambda_i^{(h)} + qG_i\} \quad (i = 1, \ldots, m) \tag{13}$$

によって λ_i を更新する．

ラグランジュ乗数を $\lambda^{(1)}, \lambda^{(2)}, \ldots$ と更新するたびにラグランジアンヒューリスティック (3.2.3 節) を行い，得られたコストが現在の z_{UB} よりも小さければ z_{UB} を更新する．

本研究では，π の制御方法として，その初期値を 2 とし，下界が 3 回連続して更新されなければ値を半分にするという基本的な方法を採用した．またラグランジュ乗数の更新回数は 80 回とする．

3.2.5 初期上界

本アルゴリズムでは欲張り法を用いて上界 z_{UB} の初期値を求める．(8) 式で定義した P^* を用いて，各辺 e_j の評価値 $eval(e_j)$ を

$$eval(e_j) = \frac{c_j}{\sum_{i \in P^*} a_{ij}} \tag{14}$$

とする．$X = \emptyset$ から始め，各反復では，$eval(e_j)$ が最小となる e_j をカットする辺として選び，$X := X \cup \{e_j\}$ とする．この処理を $P^* = \emptyset$ となるまで繰り返す．そして，上界の初期値を $z_{\mathrm{UB}} := \sum_{j:e_j \in X} c_j$ と定める．

3.2.6 初期ラグランジュ乗数

(8) 式で定義した I_j を用いて，ラグランジュ乗数の初期値を，

$$\lambda_i^{(0)} = \min_{e_j \in p_i} \frac{c_j}{|I_j|} \quad (p_i \in \hat{P}) \tag{15}$$

と定める．

3.3 経路集合 $\hat{P}$ の選択

3.3.1 初期経路集合

グラフ G から幅優先検索 (BFS) を用いて各ターミナル対の間にある経路を探す．BFS の特性により，一番初めに見つかった経路が辺数の最も少ない経路となる．その経路を $\hat{P}$ に追加し，その経路に含まれる辺を G から削除する．これを対象とするターミナル間の経路が存在しなくなるまで繰り返す．

BFS を利用する理由は，辺数の少ない経路に利用されている辺ほど，カットする辺として選ばれる可能性が高いと考えられるからである．例えば辺数が 1 の経路があればその辺は必ずカットする必要があるが，辺数が 100 であれば 100 本の辺の中から最低 1 つを選ぶだけでよい．また辺数の少ない経路に含まれる辺は，辺数の多い経路にも含まれている可能性が高いことも理由の 1 つである．

3.3.2 経路の追加

$X(\lambda^{(1)}), X(\lambda^{(2)}), \ldots, X(\lambda^{(80)})$ の中に一つでも多点対カットでないものがある場合には，以下のようにして $\hat{P}$ に経路を追加する．G' を，上述の 80 個の $X(\lambda)$ の中で，多点対カットではなく，コスト最小のものを G から除いたグラフとする．G' に対して BFS を適用し，3.3.1 節と同様のルールで $\hat{P}$ に経路を追加する．ただし，初めに見つかった経路の長さを l とすると，$l+1$ 以下の長さの経路のみを追加する．これを各ターミナル対について行う．

3.4 実行可能解の作成

$X(\lambda^{(i)})$ が多点対カットでない場合には，以下のようにして多点対カット問題の実行可能解 X を生成する．まず $X := X(\lambda^{(i)})$ とし，G から X を除いたグラフを G' とする．G' において s_i と t_i が同じ連結成分であれば，s_i と t_i に対して最小 s-t カット C を求め，そのカット辺 C を解 X に加え，G' から C を取り除く．これを各ターミナル対に対して行うことで，多点対カット X を生成する．なお，最小 s-t カット問題に対しては，多項式時間アルゴリズムが多数知られている．

3.5 冗長辺の削除

3.2.3 節，3.4 節で多点対カットを得たときには，無駄にカットした辺を戻すことで上界の改善を図る．G から多点対カット E' を除いたグラフを G' とする．現在 G' は複数の連結成分から構成されているので，各連結成分を構成する頂点の集合 $V_1, V_2, \ldots, V_d$ を求める．同時に V_i $(i = 1, 2, \ldots, d)$ に属するターミナルと対を成すターミナルが属する連結成分の添え字集合 U_i を管理しておく．

辺 $e = (v_i, v_j) \in E'$ において v_i と v_j が同じ連結成分ならば辺 e は戻すことができる．また $v_i \in V_{i'}, v_j \in V_{j'}$ において $i' \notin U_{j'}$ の時も (必ず $j' \notin U_{i'}$ であり) 辺 e を戻すことができる．ただし，後者の場合は $V_{i'}$ と $V_{j'}$ が一つの連結成分になる．そのため，$V_{i'} := V_{i'} \cup V_{j'}$ とすることで，$V_{i'}$ に $V_{j'}$ をマージする．また，$U_{i'} := U_{i'} \cup U_{j'}$ としてマージできない連結成分の情報を更新する．j' を含むような $U_{r'}$ があれば，$U_{r'} := U_{r'} \cup \{i'\}$ とする．最後に $V_{j'} := \emptyset$，$U_{j'} := \emptyset$ とする．

以上をカットした辺すべてに対して調べるが，辺を調べる順番により得られる解が異なる．本研究では全体を通して最長期間戻す辺として選ばれていない辺から調べ，戻すことが可能ならば即時戻すことにする．

3.6 アルゴリズムの流れ

アルゴリズム全体の流れは以下のようになる．

1. 初期経路集合 $\hat{P}$ を定める (3.3.1 節).
2. 乗数 $\lambda^{(0)}$ を計算し (3.2.6 節)，下界 $z_{\mathrm{LB}}(\lambda^{(0)})$ を得る (3.2.2 節).
3. SCP($\hat{P}$) の初期上界 z_{UB} を計算 (3.2.5 節).
4. (13) 式を用いてラグランジュ乗数 $\lambda^{(h)}$ から $\lambda^{(h+1)}$ を求める (3.2.4 節).
5. 下界 $z_{\mathrm{LB}}(\lambda^{(h)})$，および対応する辺集合 $x(\lambda^{(h)})$ を求める (3.2.2 節).
6. 更新した $\lambda^{(h)}$ に対する SCP($\hat{P}$) の解 $X(\lambda^{(h)})$ を求め (3.2.3 節)，多点対カットになっていれば冗長辺の削除を行う (3.5 節)．そうでなければ辺を追加して多点対カットにし (3.4 節)，冗長辺の削除を行う (3.5 節)．こうして得た解のコストが現在の z_{UB} よりも小さければ z_{UB} をその値に更新する．
7. ステップ 4 からステップ 6 を 80 回繰り返す.
8. $X(\lambda^{(1)}), \ldots, X(\lambda^{(80)})$ がすべて多点対カットになっているか，もしくは制限時間を超えれば結果を出力して終了.
9. $\hat{P}$ に経路を追加して (3.3.2 節)，ステップ 2 へ.

4 実験

本節では，3 節で提案したアルゴリズムの性能を CPLEX(10.0) と比較することで確かめる．実験環境は，CPU が Intel Xeon 3.0GHz，メモリが 8GB であり，OS は Red Hat Enterprise Linux WS release3 を用いた．また，プログラムの記述には C++ を用いた．

4.1 整数計画問題による定式化

多点対カット問題を CPLEX で解くために，多点対カット問題を混合整数計画問題に定式化して入力する．2 節の定式化を入力として与えてしまうと経路数が膨大になり解くことが困難になるため，以下

表 1　実験結果

			提案アルゴリズム			CPLEX			
$\|V\|$	$\|E\|$	k	LB	コスト	求解時間 (秒)	LB	コスト	求解時間 (秒)	総時間 (秒)
100	200	50	597	639	0.94	639	*639	1707–1829	1842.69
100	200	80	656	754	9.91	692	768	3543–3788	–
100	400	50	1753	1951	83.44	1793	1938	17377	–
100	400	80	1962	2307	1051.00	1993	2505	6653–13125	–
100	2000	50	11860	12154	61.71	12108	12154	14919	–
100	2000	80	13034	14278	1456.92	13176	18858	2845–18000	–
200	400	50	1046	1088	179.84	1088	*1088	636	16906.3
200	400	80	1168	1343	66.91	1192	1448	2241	–
200	400	120	1320	1588	782.65	1343	1730	4449	–
200	800	50	2632	2688	59.57	2688	*2688	3381	3503.19
200	800	80	2917	3255	315.95	2967	4497	$\geq$ 5345	–
200	800	120	3146	4158	460.93	3200	5786	$\geq$10472	–
400	800	50	13375	1405	863.22	1405	*1405	2831–5732	5749.19
400	800	100	1659	2083	1367.98	1670	3156	$\geq$5459	–
400	800	150	1775	2582	419.26	1832	3352	$\geq$12431	–
400	1600	50	3802	3807	163.50	3807	*3807	3145	3145.23
400	1600	100	5117	5770	561.49	5254	–	–	–
400	1600	150	5410	7155	384.37	5598	–	–	–

の定式化を用いる．

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} c_j x_j \\
\text{s.t.} \quad & y^r_{s_r} \geq y^r_{t_r} + 1 && (r = 1, 2, \ldots, k) \\
& y^r_{\mathrm{head}[j]} \leq y^r_{\mathrm{tail}[j]} + x_j && (j = 1, 2, \ldots, n,\ r = 1, 2, \ldots, k) \\
& y^r_{\mathrm{tail}[j]} \leq y^r_{\mathrm{head}[j]} + x_j && (j = 1, 2, \ldots, n,\ r = 1, 2, \ldots, k) \\
& x_j \in \{0, 1\}, y^r_j \geq 0
\end{aligned}
$$

x_j は辺 e_j をカットするときは値 1，しないときは値 0 をとる 0-1 変数である．head$[j]$ と tail$[j]$ は辺 e_j の 2 つの端点を表す．y^r_i は r 番目のターミナル対に対する頂点 i のポテンシャルを表す．制約の第 2 式と第 3 式より $x_j = 0$ ならば $y^r_{\mathrm{head}[j]} = y^r_{\mathrm{tail}[j]}$ なので，各 r に対し，ひとつの連結成分に含まれる頂点の y^r_i の値は等しくなければならない．一方，制約の第 1 式は s_r と t_r のポテンシャルが 1 以上異なることを要求しているため，この定式化における実行可能解に対応するカットにおいて，s_r と t_r が異なる連結成分に属することが保証されている．

4.2　入力グラフと結果

入力グラフとして頂点数が 100，200，400 のグラフをランダムに作成した．作成方法は指定した辺数だけランダムに 2 頂点を選び，辺で結ぶ方法である．非連結な頂点対を優先的に選んで辺で結ぶことで，連結なグラフを作成している．各辺のコストは，1 から 30 までの整数から一様にランダムに選んだ．提案アルゴリズムの制限時間を 1800 秒，CPLEX の制限時間を 18,000 秒とした．結果を表 1 に示す．$|V|$ は頂点数を，$|E|$ は辺数を，k はターミナル対の数を示し，制限時間内に得られた解の中で最もコストの小さいものと，その解を得るまでにアルゴリズムが費やした時間 (求解時間)，及び得られた中で最大の下界 (LB) を記す．CPLEX においては正確な時間を確認できなかったため，表示から推定されるおよその時間を記す．18,000 秒以内に厳密な最適解であることを確認できたものはコストの左部分に '*' をつ

け，総時間を記す．時間内に実行可能解が得られなかった場合は '–' とした．

表 1 より，ターミナル対の数やグラフのサイズが大きくなると CPLEX では問題を解くことが困難であることがわかる．また，多くの問題例に対し，提案アルゴリズムは CPLEX より低いコストの解をより少ない時間で得ることができた．同じコストの解が得られた場合においても，より少ない時間で解を得ることが確認できた．

5 終わりに

本稿では，無向グラフにおける多点対カット問題に対し，集合被覆問題による定式化に基づくアルゴリズムを提案した．CPLEX を比較対象とした計算実験の結果，提案アルゴリズムが平均的により良い結果を与えることが確認できた．

参考文献

[1] A. Caprara, M. Fischetti, P. Toth, A heuristic method for the set covering problem, Operations Research, Vol. 47 (1999), pp. 730–743

[2] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, M. Yannakakis, The complexity of multiway cuts, Proceedings of the 24th Annual ACM Symposium on Theory of Computing (1992), pp. 241–251

[3] M. L. Fisher, The Lagrangian relaxation method for solving integer programming problems, Management Science 27 (1981), pp. 1–18

[4] N. Garg, V. V. Vazirani, M. Yannakakis, Approximate max-flow min-(multi)cut theorems and their applications, Proceedings of the 25th Annual ACM Symposium on Theory of Computing (1993), pp. 698–707

[5] M. Held, R. M. Karp, The traveling salesman problem and minimum spanning trees, Part II, Mathematical Programming 1 (1971), pp. 6–25

[6] V. V. Vazirani, Approximatioin Algorithms, Springer (2001)

[7] M. Yannakakis, P. C. Kanellakis, S. C. Cosmadakis, and C. H. Papadimitriou, Cutting and partitioning a graph after a fixed pattern, Proceedings of the 10th Colloquium on Automata, Languages and Programming (1983), pp. 712–722