情報処理学会論文誌 Vol.57 No.10 2260–2271 (Oct. 2016)

連続型・時間/費用トレードオフ最小全域木問題

片岡 靖詞[1,a)] 村崎 暢彦[2]

受付日 2016年1月12日, 採録日 2016年6月2日

概要：最小全域木問題は，グラフの各枝に時間などのデータが与えられているとき，総時間を最小化する全域木を求める問題である．本研究では，各枝に与えられた時間が，資金を投入することにより短縮できる場合を考え，限られた資金の中で総時間を最小にする全域木を求める問題を扱う．各枝における時間の短縮は，資金の投入額に応じて連続的かつ線形的に時間短縮ができる場合を想定しているので，この問題を連続型時間/費用トレードオフ最小全域木問題と呼ぶことにする．このような問題設定は，プロジェクトスケジューリングでは，CPM として古くから知られている扱いやすい問題設定であるが，最小木問題の場合は，線形的に時間短縮ができる場合でも NP 困難である．また，解には特徴的な構造を持つことが知られているが，この性質について先行研究とは別の証明を示す．定式化の構造からラグランジュ緩和は 2 つの独立した扱いやすい問題に分割でき，効果的に上下界値が導出できる．また，この上下界値を利用して釘付けテストを行い，多くの場合において分枝限定法に入る前に最適解が得られることを示す．釘付けテストは枝数が多くなると，時間的負担も増えてくるものの，疎なグラフ，特に平面的グラフにおいては，先行研究よりも優れた結果を得た．

キーワード：時間/費用トレードオフ関係，最小木問題，ラグランジュ緩和，釘付けテスト，分枝限定法

Continuous Time/Cost Tradeoff Minimum Spanning Tree Problem

Seiji Kataoka[1,a)] Masahiko Murasaki[2]

Received: January 12, 2016, Accepted: June 2, 2016

Abstract: Given an undirected graph and a parameter such as time on each edge, the minimum spanning tree problem is a problem of finding a spanning tree that minimizes total time. In this study, we consider the case that the time on each edge can be reduced by throwing resource such as extra fare into the edge, and find the minimum total time spanning tree within a total resource constraint. As for the reduction of time for each edge, we assume that it can be continuously and linearly reduced. Therefore, we call the problem *the continuous time/cost tradeoff minimum spanning tree problem*. This kind of continuous and linear tradeoff relationship is often seen in the subject of project scheduling problems, which has been studied before as CPM and is easily solvable. But in the case of the minimum spanning tree problem, even though the tradeoff relationship is continuous and linear, the problem is known to be NP-hard. Also we prove that the optimal solution has a characteristic structure. These proofs are different from those seen in the past research. Based on our formulation, we show that the Lagrangian relaxation problem is divided into two independent tractable problems. Hence, we succeed in obtaining upper and lower bounds efficiently. By applying these bounds into the pegging test, we show that in many cases optimal solutions are obtained before going forward to the branch-and-bound algorithm. As the number of edges becomes large, the computational burden increases, but our computational results are superior to the results in the past research, especially when the graph is sparse.

Keywords: time/cost tradeoff, minimum spanning tree, Lagrangian relaxation, pegging test, branch-and-bound algorithm

1. はじめに

最小全域木問題は，グラフ・ネットワークの基本的な問

[1] 防衛大学校情報工学科
Department of Computer Science, National Defense Academy, Yokosuka, Kanagawa 239–8686, Japan
[2] 陸上自衛隊
Japan Grand Self Defense Force
[a)] seiji@nda.ac.jp

題であるだけでなく，より複雑な問題の部分問題として，また巡回路問題の緩和問題としても表れ，通信網や VLSI の設計などにも応用例が多い [2], [3], [17]．それらの問題では各枝に費用などのデータが与えられるが，現実には別の要因などがともなうことがあるため，事前に確定した費用が与えられるとは限らない．たとえば，枝として通信ケーブルを設置する場合，設置方法により通信リスクがともなうことがある．直線的にケーブル張ることができないような場所では，柱を立てたり床下や天井裏に収めたりするなど費用をかければ通信リスクを軽減できるが，逆に地形や建物の形状に合わせてむき出し状態でケーブルを張るなどすると安上がりだが通信リスクは高くなる．また折衷案も何種類か考えられる．これらのケーブルの設置費用と通信リスクとはトレードオフの関係にあり，本研究では限られた総予算の中で，総通信リスクを最も小さくするような全域木を求める問題を考える．

2 種類のデータがトレードオフの関係にある問題設定は，プロジェクトスケジューリングの分野においては PERT・CPM として古くから研究されており [14], [15]，そして時間/費用トレードオフという用語を含む問題群もいくつも知られており，Değirmenci-Azizoğlu [5] では，それらの問題群を系統的に分類している．トレードオフ関係というのは，前述の例におけるリスクと費用，プロジェクトスケジューリングにおける時間と費用，そのほかにも考えられるが，本論文ではこの問題のことを，多くの研究があるプロジェクトスケジューリングでの呼称に準じて，**時間/費用トレードオフ最小全域木問題**（Time/Cost trade-off Minimum Spanning Tree: T/CMST）と呼ぶことにする．

さらに PERT・CPM では，投入する費用に応じていくつかの可能な時間短縮案から 1 つを選ぶという離散型モデルと，投入費用に応じて線形的に時間短縮ができる連続型モデルとに大別している．そして，離散型モデルは NP 困難であるのに対し [4], [5], [6]，連続型モデルは簡単に解けることが古くから示されている [11], [14], [15]．一方，最小全域木では，Kataoka-Yamada [9]（以下 KY と略記）が，各点間に標準か特急かの 2 種類のモードを想定し，これらの二者択一とする離散型モデルと，投入資金に応じて標準と特急の間において線形的に時間短縮ができる連続型モデルの両方を扱っており，両者とも NP 困難であることを示している．このことは，プロジェクトスケジューリングと最小全域木とは，本質的な違いがあることを意味している．これらのうち本論文では連続型モデルを扱っており，以降では，連続型（Continuous）時間/費用トレードオフ最小全域木問題という位置づけで，C-T/CMST 呼ぶことにする．

関連研究としては，各枝に時間と費用が与えられて一方を目的関数，他方をナップサック型制約として考える問題（knapsack constrained maximum spanning tree problem [16]）がある．この問題には the minimum spanning tree problem subject to side constraint [1] など名称の異なるものがいくつか存在するが，いずれも各枝に 2 種類のデータはあるものの，本研究で扱う C-T/CMST のように複数のモードという概念はない．ただし，2.3 節で説明するように C-T/CMST の最適解は，ある枝を選択することで，各枝の採るべきモードも決まるという特殊な構造を持つため，選択する枝を変えながら knapsack constrained MST を繰り返し解くことで C-T/CMST も解くことが可能である．KY のアルゴリズムはこの考え方に基づくものである．さらに，resource constrained MST [7] という問題もあり，これはナップサック制約に相当する資源制約が複数あり，それらを満足する最小木を求めるものである．これらの資源制約を目的関数に持っていったものとして，multi-objective MST [13] というものもあり，複数ある目的関数値の min-max あるいは max-min などを求める問題である．しかし，いずれも 1 つの枝に複数のデータが割り当てられている点で類似はしているものの，モードという概念もなく，直接的に C-T/CMST の効率的なアルゴリズム開発につながる可能性は低い．

本研究では KY で提示している問題のうち連続型を扱うが，定式化を明示し，ラグランジュ緩和した問題が，2 つの独立した扱いやすい問題に分割できることを示し，効果的に上下界値を得る方法を示す．そして，この上下界値をもとにした釘付けテストを提案する．釘付けテストにおいて決定できなかった部分においては，最適解を得るための分枝限定法を提案する．

本論文は 2 章で定式化を示すが，定式化とソルバの活用だけでは解くことが困難であることを確かめる．また，定式化の変形により，KY でも示している解の特徴的な構造の別証とこの構造が上界値算法につながることを示す．3 章ではラグランジュ緩和とラグランジュ双対問題および釘付けテストを説明し，4 章で分枝限定法を示す．5 章で計算機実験の結果を示し，KY とまったく同じ例題も用いて，本研究で示すアルゴリズムの優越性を示す．

2. 定式化および問題の特性

2.1 定式化

無向グラフ $G = (E, V)$ において，枝 $e = (i, j)$ には標準所要時間 t_e^s と特急所要時間 t_e^r が与えられている．標準所要時間を達成するには費用 c_e^s がかかり，特急所要時間を達成するためには費用 c_e^r がかかるものとする．また，これらの所要時間と費用とがトレードオフ関係になるよう，式 (1) が成り立っていることを仮定する（**図 1**）．

$$c_e^s \le c_e^r, \quad t_e^s \ge t_e^r \qquad \forall e \in E \tag{1}$$

連続型モデルでは，所要時間は投入する費用に従って，図 1 の線分上で連続的に短縮できる．特に両端の (c_e^s, t_e^s) を**標準計画**，(c_e^r, t_e^r) を**特急計画**と呼ぶことにする．定義さ

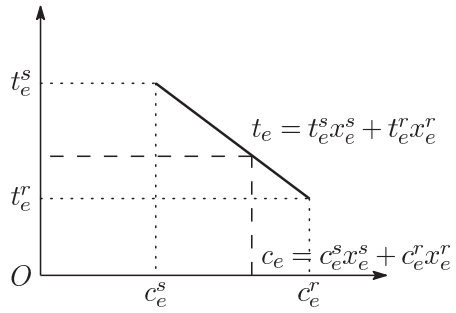

図 1 時間/費用トレードオフ関係
Fig. 1 The relationship of time/cost tradeoff.

れた区間内での費用 c_e と所要時間 t_e の表現方法はいくつか考えられるが，本論文では変数 x_e^s，x_e^r を用い，標準計画と特急計画の凸結合として表すことを考える．

$$\begin{aligned} c_e &= c_e^s x_e^s + c_e^r x_e^r \\ t_e &= t_e^s x_e^s + t_e^r x_e^r \\ x_e^s + x_e^r &= 1, \quad x_e^s, x_e^r \geq 0 \end{aligned} \tag{2}$$

また，解となる全域木の枝として採る/採らないを決める 0-1 変数を y_e（$e \in E$）とする．このとき，C-T/CMST は式 (3)–(8) のように定式化できる．

$$\min \sum_{e \in E} t_e^s x_e^s + \sum_{e \in E} t_e^r x_e^r \tag{3}$$

$$\text{s.t.} \sum_{e \in E} c_e^s x_e^s + \sum_{e \in E} c_e^r x_e^r \leq B \tag{4}$$

$$\sum_{e \in E} y_e = n - 1 \tag{5}$$

$$\sum_{e \in E(S)} y_e \leq |S| - 1 \quad \forall S \subset V \tag{6}$$

$$x_e^s + x_e^r - y_e = 0 \quad \forall e \in E \tag{7}$$

$$x_e^s, x_e^r \geq 0, \quad y_e \in \{0, 1\} \quad \forall e \in E \tag{8}$$

式 (3) は目的関数で，総時間の最小化を示す．式 (4) は総費用が予算を超えないことを示す．点の部分集合を S（$\subset V$）とし，$E(S)$ を両端の点が S に属する枝集合とすると，式 (5) と式 (6) をあわせて全域木であることを示し，特に式 (6) は部分巡回路除去制約と呼ばれる．ここまで式 (3)，(4) と式 (5)，(6) は構成している変数が独立していることに注意する．これらの変数を結び付けているのが式 (7) であり，全域木の枝として採られるときには，$x_e^s + x_e^r = 1$ となり，式 (8) の非負条件より，所要時間 t_e や費用 c_e が式 (2) に示すように標準計画と特急計画の凸結合になっていることを示す．また，全域木の枝として採られないときは，$x_e^s + x_e^r = 0$，つまり $x_e^s = x_e^r = 0$ となり，その枝には所要時間も費用もかからないようになる．

2.2 ソルバの活用

定式化が記述できれば，ソルバなどによって解決できる問題も少なくない．しかし，C-T/CMST は小さな問題でも部分巡回路除去制約の数が膨大になるため，定式化どおりソルバに入力することは事実上困難である．そこで，本研究で用いる商用ソルバである Gurobi [8] の callback 機能の活用を試みた．callback 機能とは，部分巡回路除去制約 (6) を除いて入力し，Gurobi 内の分枝限定法の途中で実行可能解が出たとき（変数 y_e（$e \in E$）がすべて 0-1 解を得たとき）に部分巡回路の有無をチェックし，部分巡回路ができていれば，それを除去する制約を逐次追加していくことで，最終的に最適解を得る手法である．

ここでは，式 (6) に加えて，カット型制約とも呼ばれる式 (9) も考慮する．それは，Gurobi 内の分枝限定法で整数解が得られたき，通常いくつかの連結成分に分解されており，そのうちのある連結成分を構成する点集合を S とし，一方の点が S に属し他方の点が S に属さないカットとなる枝集合を $C(S)$ とするとき，次のような制約式のことである．

$$\sum_{e \in C(S)} y_e \geq 1 \tag{9}$$

ある整数解において，部分巡回路や連結成分は 1 つでないため，追加できる制約も 1 本だけではない．ここでは，その整数解から見つけられるものすべてを入れることにする．この方法だと，ある整数解から制約を見つける手間と追加する制約数は増えるが，通常の最小全域木問題の場合には，全体として高速に解けることが多いという経験に基づいている．

この予備的実験に用いるグラフと計算機については 5 章で詳細を説明するが，P_n^m というのは，点の数 n，枝の数 m の平面的グラフのことであり，時間や費用を [1,1000] の一様乱数で発生させ，式 (1) の仮定を満足するように調整した 10 問の平均値を**表 1** に示した．表 1 において，B は予算制約，cpu は計算時間（秒），そして cuts は callback 機能のにおいて追加した制約の数である．

表 1 Gurobi の callback 機能を用いた結果
Table 1 The results of using Gurobi callback.

	B	cpu	cuts
P_{20}^{46}	8000	0.03	19.2
P_{30}^{74}	12000	0.17	221.6
P_{50}^{127}	20000	2.97	1285.0
P_{100}^{260}	40000	—	25371.3

表中 “—” は 1,200 秒で解けなかったことを示すが，表 1 から分かるように，点の数が 20 や 30 のごく小さな例題であれば，ソルバを用いるだけでも難なく解くことができる．しかし点の数が 50 になると解き難さが目立つようになり，P_{100}^{260} では 1,200 秒のうちに 25,000 本もの制約を追加しても，10 問中 1 問も結果が得られなかった．後ほど示すが，KY のアルゴリズムでも，P_{100}^{260} では数秒で最適解を得ている．

本研究では P_{100}^{260} 程度の例題を最も小さなサイズととらえており，これ以上のサイズの問題を KY のアルゴリズムよりもさらに高速に解くためのアルゴリズムを紹介する．

2.3 解の構造

KY では，C-T/CMST の最適解の構造として，たかだか 1 本の枝を除いて，標準計画または特急計画のいずれか一方を採り，標準計画と特急計画の間になるのはたかだか 1 本の枝だけであり，そのときの総費用はちょうど B になることを示している．彼らは，2 本以上の枝において標準と特急の間になる解を仮定し，標準と特急の間になる枝がより少ない本数でより良い解を構築していく方針で証明をしているが，この性質も定式化の変形により別証を与えることができる．

定理 1 C-T/CMST の最適解は，たかだか 1 本の枝を除いて標準計画または特急計画のいずれか一方を採り，標準計画と特急計画の間になるのはたかだか 1 本の枝だけであり，そのときの総費用はちょうど B になる．ただし，すべて特急計画で総時間最小の全域木の総費用が B よりも小さいときは，この全域木が最適解となる．これは予算が十分にあることを意味しており，このときは総費用は B にはならない．

証明 枝の部分集合を T（$\subset E$）とする（C-T/CMST の場合は，T として最適全域木の枝集合を想定しているが，この証明では T は全域木を構成する枝である必要はなく，式 (5)，(6) を満足している必要もない）．T に属する枝のみを対象とした所要時間と費用の振り分けが問題なので，$y_e = 1$（$e \in T$），$y_e = 0$（$e \notin T$）とする．また式 (5)，(6) は目的関数とは無関係なので除去し，式 (7) の対象を T の枝に限定すると次のようになる．

$$\min \quad \sum_{e\in T} t_e^s x_e^s + \sum_{e\in T} t_e^r x_e^r \tag{10}$$

$$\text{s.t.} \quad \sum_{e\in T} c_e^s x_e^s + \sum_{e\in T} c_e^r x_e^r \le B \tag{11}$$

$$x_e^s + x_e^r = 1 \qquad \forall e \in T \tag{12}$$

$$x_e^s, x_e^r \ge 0 \qquad \forall e \in T \tag{13}$$

式 (12)，(13) より，$x_e^s = 1 - x_e^r \le 1$ であり，これを式 (10)，(11) に代入すると，式 (14) のような連続型ナップサック問題になる．この問題の最適解は貪欲戦略で得られる [10], [12]．ただし，仮定 (1) より $t_e^r - t_e^s \le 0$ であり，目的関数は最小化になっていることに注意する．

$$\begin{aligned} \min \quad & \sum_{e\in T}(t_e^r - t_e^s)x_e^r + \sum_{e\in T} t_e^s \\ \text{s.t.} \quad & \sum_{e\in T}(c_e^r - c_e^s)x_e^r \le B - \sum_{e\in T} c_e^s \\ & 0 \le x_e^r \le 1 \end{aligned} \tag{14}$$

標準計画と特急計画の間になりうる 1 本の枝 f は，T に属する枝を

$$\frac{t_{e_1}^r - t_{e_1}^s}{c_{e_1}^r - c_{e_1}^s} \le \frac{t_{e_2}^r - t_{e_2}^s}{c_{e_2}^r - c_{e_2}^s} \qquad e_1 < e_2, \quad e_1, e_2 \in T \tag{15}$$

となるように枝番号を付け替えたとき，

$$\sum_{e=1}^{f-1}(c_e^r - c_e^s) \le B - \sum_{e\in T} c_e^s < \sum_{e=1}^{f}(c_e^r - c_e^s) \tag{16}$$

となる枝を f と定める．このとき $e = 1, 2, \cdots, f-1$ に対しては，$x_e^r = 1$ つまり特急計画として費用 c_e^r を投入し，$e = f+1, f+2, \cdots, |E|$ に対しては，$x_e^r = 0$ つまり標準計画として費用 c_e^s を投入する．枝 f に関しては，残りの予算が残らないように使うようにするため，費用 $c_f^s + B - \sum_{e\in T} c_e^s - \sum_{e=1}^{f-1}(c_e^r - c_e^s) = B - \sum_{e=1}^{f-1} c_e^r - \sum_{e=f+1}^{|T|} c_e^s$ を投入し，解は式 (17) で与えることができる．

$$x_f^r = \left(B - \sum_{e\in T} c_e^s - \sum_{e=1}^{f-1}(c_e^r - c_e^s)\right) \Big/ (c_f^r - c_f^s) \tag{17}$$

また，定理の後半部分は明らかである．

□

直感的な理解としては，$(t_e^r - t_e^s)/(c_e^r - c_e^s)$ は図 1 の線分の傾きを示しているので，仮定 (1) より傾きが負であることに注意して，

$$\frac{t_e^r - t_e^s}{c_e^r - c_e^s} < \frac{t_f^r - t_f^s}{c_f^r - c_f^s} \implies x_e^s = 0, x_e^r = 1 \tag{18}$$

$$\frac{t_e^r - t_e^s}{c_e^r - c_e^s} > \frac{t_f^r - t_f^s}{c_f^r - c_f^s} \implies x_e^s = 1, x_e^r = 0 \tag{19}$$

のように設定している．つまり枝 f よりも傾きが小さい（急勾配：対費用効果が高い）ときには特急計画を選び，傾きが大きい（緩い勾配：対費用効果が低い）ときには標準計画を選ぶ．枝 f だけが中間的な値をとり，全体としてちょうど予算 B をいっぱいに使いきるように残った予算を投入する．

この性質は，ある木構造の枝 T が与えられたときの最適解，つまりその木を解とする最良の上界値を得るために使われ，アルゴリズム $CalcUB(T)$ でその値を求めることができる．

3. 上下界値と釘付けテスト

3.1 ラグランジュ緩和

定式化を構成している変数のタイプには，標準計画・特急計画（および中間計画）を決める $\boldsymbol{x}$-型の変数と，全域木を形作る $\boldsymbol{y}$-型の変数とがあり，両者を結び付けているのは式 (7) だけである．したがって，式 (7) をラグランジュ緩和すれば，緩和問題は，$\boldsymbol{x}$-型の変数だけで構成される問題と，$\boldsymbol{y}$-型の変数だけで構成される問題とに分割できる．

ラグランジュ緩和する前に，元の定式化に対し (i) 式 (4) を等式にする，(ii) $x_e^s + x_e^r \le 1$（$e \in E$）を追加する，(iii) $\sum_{e\in E} x_e^s + \sum_{e\in E} x_e^r = n - 1$ を追加する，これらの操作を施しても，（B がすべて特急計画で実行できる総費用より

Algorithm CalcUB(T)

Input: a set of edges T, which is expected to be a tree.
Return: the greedily assigned value, which gives an upper Bound when T is a tree.

```
c_e := c_e^s (e ∈ T), UB := Σ_{e∈T} t_e^s, b := B − Σ_{e∈T} c_e^s
if (b < 0) then infeasible return // shortage of budget
Sort (t_e^r − t_e^s)/(c_e^r − c_e^s) (e ∈ T) in non-decreasing order.
e := 1
while e ≤ |T| and (c_e^r − c_e^s ≤ b) do
  c_e := c_e^r
  UB := UB + (t_e^r − t_e^s)
  b := b − (c_e^r − c_e^s)
  e := e + 1
end while
if e ≤ |T| then
  f := e, c_f := c_f^s + b
  UB := UB + b(t_f^r − t_f^s)/(c_f^r − c_f^s)
end if
return UB
```

も大きくなければ）実行可能解を切り落とすことはない．この操作の後，式 (7) に（見栄えのために両辺 -1 倍して）ラグランジュ乗数 λ_e を掛けて緩和すれば，$\boldsymbol{x}$-型の変数だけでできる問題 (20)

$$\begin{aligned} L_x(\boldsymbol{\lambda}) = \min & \sum_{e\in E}(t_e^s-\lambda_e)x_e^s + \sum_{e\in E}(t_e^r-\lambda_e)x_e^r \\ \text{s.t.} & \sum_{e\in E} c_e^s x_e^s + \sum_{e\in E} c_e^r x_e^r = B \\ & x_e^s \quad + \quad x_e^r \le 1 \quad \forall e \in E \\ & \sum x_e^s \ + \ \sum x_e^r = n-1 \\ & x_e^s, x_e^r \ge 0 \end{aligned} \tag{20}$$

と，$\boldsymbol{y}$-型の変数だけでできる最小全域木問題 (21)

$$L_y(\boldsymbol{\lambda}) = \min\left\{\sum \lambda_e y_e | \boldsymbol{y} \text{ is spanning tree}\right\} \tag{21}$$

とに分離でき，下界値は式 (22) で得られる．

$$L_x(\boldsymbol{\lambda}) + L_y(\boldsymbol{\lambda}) \tag{22}$$

分割されたそれぞれの問題は規模が小さくなるだけでなく，最適解を得るために $L_x(\boldsymbol{\lambda})$ に対しては Gurobi [8] などの商用ソフトを活用し，$L_y(\boldsymbol{\lambda})$ に対しては Prim などの最小木アルゴリズム [2], [17] を適用すれば，簡単に求めることができる．また，ラグランジュ緩和する前に施した操作 (i)，(ii)，(iii) により，$\boldsymbol{\lambda} = \boldsymbol{0}$ のときは，元の定式化の$\boldsymbol{y}$-型変数を線形緩和したものと等価になるので，下界値 (22) は線形緩和以上の値を与えることが保証できる．さらに，$L_y(\boldsymbol{\lambda})$ は最小木問題であるため，何らかの全域木 T を返す．その T を用いてアルゴリズム $CalcUB(T)$ を適用すれば，実行可能解も得られる．3.2 節のラグランジュ双対問題の過程において，$L_y(\boldsymbol{\lambda})$ を解く機会は頻繁にあるので，その途中で最適解を与える全域木が得られる可能性は十分期待できる．

3.2 ラグランジュ双対問題

最大の下界値を与えるラグランジュ双対問題とは，式 (22) の最大値を求めることであり，式 (23) のように書くことができる．

$$\max_{\boldsymbol{\lambda}}\{L_x(\boldsymbol{\lambda}) + L_y(\boldsymbol{\lambda})\} \tag{23}$$

式 (22) は区分線形凹関数になることが知られており [2]，最大の下界値を与える $\boldsymbol{\lambda}^L$ を求めるために劣勾配法を用いる．劣勾配法は初期のラグランジュ乗数を $\boldsymbol{\lambda}^0$ を与え，式 (24) に従って逐次更新する方法である．ここで $\boldsymbol{d}^\ell$，θ^ℓ は ℓ 回目の繰返しにおける更新方向および更新幅である．

$$\boldsymbol{\lambda}^{\ell+1} = \boldsymbol{\lambda}^\ell + \boldsymbol{d}^\ell\theta^\ell \tag{24}$$

更新方向 $\boldsymbol{d}^\ell$ は，通常式 (25) のように定める [2]．ここで $\boldsymbol{x}^\ell$ は，ある $\boldsymbol{\lambda}^\ell$ における $L_x(\boldsymbol{\lambda}^\ell)$ の最適解を $(x_e^s)^\ell$，$(x_e^r)^\ell$ $(e \in E)$ としたとき e 番目の要素が $(x_e^s)^\ell + (x_e^r)^\ell$ であるようなベクトルであり，また $\boldsymbol{y}^\ell$ は $L_y(\boldsymbol{\lambda}^\ell)$ の最適解である．したがって，更新方向は，式 (7) の左辺ベクトルから得られる．

$$\boldsymbol{d}^\ell = -\boldsymbol{x}^\ell + \boldsymbol{y}^\ell \tag{25}$$

更新幅 θ^ℓ も，通常式 (26) のように定める．

$$\theta^\ell = \rho \cdot \frac{UB - (L_x(\boldsymbol{\lambda}^\ell) + L_y(\boldsymbol{\lambda}^\ell))}{\sum_{e\in E}(d_e^\ell)^2} \tag{26}$$

ρ は $0 < \rho < 2$ の値を用いることが推奨されているが，大きく設定しすぎると収束が遅くなり，小さく設定しすぎると下界値の上昇が遅くなるといわれている．したがって，本研究では初期値を 1.5 とし，下界値の更新がない繰返しが 5 回連続すると ρ を 0.8 倍した．

劣勾配法の繰返しは 100 回を基本としたが，100 回の繰返しを終えたときにまだ十分下界値が上昇していない，つまり上下界値の差が 1 以上であるときには劣勾配法を続ける．ただし，100 回の繰返しが終わったときには，ρ の値は小さくなりすぎている場合が多いので，再び $\rho := 1.5$ に設定し直すことにした．つまり，100 回繰り返した後の $\boldsymbol{\lambda}^{100}$ を次の初期値とすることを意味する．これを最大 5 回繰り返すことにした．

上下界の差が 1 未満になると，十分な下界値が出たものと判断することにしたが，劣勾配法の繰返しが 70 回に達していないときには，さらに下界値の上昇が期待できるので，上下界値の差が早い段階で 1 未満になったとしても 70 回までは繰り返すことにした．また，等式制約 (7) を緩和しているので，途中で $\boldsymbol{x}^\ell = \boldsymbol{y}^\ell$ となったら最適解が得られたことを意味する．以上の諸設定を含めたアルゴリズムが，Lagrangian Dual である．

Algorithm Lagrangian Dual

Input: Nothing
Return: lower bound $BestLB$, upper bound $BestUB$ and the optimal $\boldsymbol{\lambda}^L$
$\ell := 0$, $\lambda_e^\ell := 0 (e \in E)$, $\rho := 1.5$
while $\ell < 500$ **do**
  if $(100|\ell)$ **then** $\rho := 1.5$
  Solve $L_x(\boldsymbol{\lambda}^\ell)$ by Gurobi
  Solve $L_y(\boldsymbol{\lambda}^\ell)$ by Prim's MST algorithm
  $LB := L_x(\boldsymbol{\lambda}^\ell) + L_y(\boldsymbol{\lambda}^\ell)$;
  if $(LB > BestLB)$ **then** $BestLB := LB$ and $\boldsymbol{\lambda}^L := \boldsymbol{\lambda}^\ell$
  $UB :=$CalcUB(T) where $T := \{e | y_e^\ell = 1, e \in E\}$
  if $(UB < BestUB)$ **then** $BestUB := UB$
  if $BestLB = BestUB$ **then return** //optimal solution
  if $BestUB - BestLB < 1.0$ and $\ell > 70$ **then return**
  $\boldsymbol{d}^\ell := -\boldsymbol{x}^\ell + \boldsymbol{y}^\ell$ // by (25)
  if $BestLB$ is not updated five times in a row
  then $\rho := 0.8\rho$.
  $\theta^\ell := \rho(BestUB - LB) / \sum_{e \in E} (d_e^\ell)^2$ // by (26)
  $\boldsymbol{\lambda}^{\ell+1} := \boldsymbol{\lambda}^\ell + \theta^\ell \boldsymbol{d}^\ell$ // by (24)
  $\ell := \ell + 1$
end while

3.3 釘付けテスト

釘付けテストの基本的な考え方は，ある枝を必ず採る/採らないと固定して求めた下界値が，上界値を超えるときは，その固定からは最適解が得られる見込みはなく，対象の枝を採らない/採ると固定するものである．釘付けテストは，分枝限定法のどの分枝頂点においても適用可能ではあるが，時間的負担も大きくなるので，分枝限定法に入る前処理として十分なだけ枝を固定し，問題の縮小を図るのが通常である．

離散型の T/CMST では，各枝に対してどのモードで枝を採るか・採らないかを決めなければならないが，連続型では，採る枝集合が決まりさえすれば，それらが標準計画になるか特急計画（あるいは中間計画）になるかは，CalcUB(T) に従って簡単に決められる．したがって，各枝に対して採る・採らない・未固定を決めるだけでよい．

アルゴリズム Pegging Test の入力 $\boldsymbol{\lambda}^L$ は，ラグランジュ双対問題を解いた結果，最大の下界値を与えるラグランジュ乗数である．$\boldsymbol{x}^L$，$\boldsymbol{y}^L$ も 3.2 節と同様に定義する．

実際に下界値を計算をするときは，$x_e^L = 0(1)$ のときの $L_x(\boldsymbol{\lambda}^L | x_e^L = 0(1))$ や $y_e^L = 0(1)$ のときの $L_y(\boldsymbol{\lambda}^L | y_e^L = 0(1))$ は求める必要はなく，ラグランジュ双対問題を解く際に得ていた $LB_x := L_x(\boldsymbol{\lambda}^L)$，$LB_y := L_y(\boldsymbol{\lambda}^L)$ を用いる．また，$x_e^L = 1(0)$ のときの $L_x(\boldsymbol{\lambda}^L | x_e^L = 0(1))$ は，Gurobi の中で $x_e^s + x_e^r = 0(1)$ に設定して求める．

一方，$\boldsymbol{y}^L$ は木構造 $T^L := \{e | y_e^L = 1, e \in E\}$ を構成しているので，$y_e^L = 0$ のときに $L_y(\boldsymbol{\lambda}^L | y_e^L = 1)$ を求めるためには，$T^L \cup \{e\}$ でできる閉路の枝から e を除く最大の λ_e^L を持つ枝 e' を探し，$\lambda_e^L - \lambda_{e'}^L$ を下界値の増分とすることができる．また，$y_e^L = 1$ のときに $L_y(\boldsymbol{\lambda}^L | y_e^L = 0)$ を求めるためには，$T^L \setminus \{e\}$ で定義されるカットから e を除く最小の λ_e^L を持つ枝 e'' を探し，$\lambda_{e''}^L - \lambda_e^L$ を下界値の増分とすることができる．しかしながら，グラフの枝密度が高くなると e'' を見つける手間も複雑になり，計算効率も劣化するため，5 章で示す計算機実験では $\lambda_e = \infty$ として Prim のアルゴリズムを用いて最小全域木を求めている．

Algorithm Pegging Test

Input: $BestUB$, $\boldsymbol{\lambda}^L$
Return: F_1, F_0: the sets of edges fixed to 1, 0, respectively
for each $e \in E$ **do**
  solve $LB_x := L_x(\boldsymbol{\lambda}^L | x_e^L = 0)$ and $LB_y := L_y(\boldsymbol{\lambda}^L | y_e^L = 0)$
  if $LB_x + LB_y > BestUB$
  then Edge e is included in the optimal tree.
  solve $LB_x := L_x(\boldsymbol{\lambda}^L | x_e^L = 1)$ and $LB_y := L_y(\boldsymbol{\lambda}^L | y_e^L = 1)$
  if $LB_x + LB_y > BestUB$
  then Edge e is excluded from the optimal tree.
  else PegShortage[e] $:= BestUB - (LB_x + LB_y)$
end for

枝 e を採ると固定した問題を解き，釘付けができなかったとき $BestUB - (LB_x + LB_y)$ の値を PegShortage[e] として記憶しておく．これは枝 e を採らないと確定するためには，下界値としての不足度合を示している．この値は次節の分枝限定法の際に，分枝変数の選択基準として用いる．

4. 分枝限定法

分枝限定法は，枝を採る枝/採らない枝集合を制限した子問題を解きながら，下界値の情報を用いて最適解の存在する範囲を限定し，最終的に厳密解を得ようとするものである．ここで，ある子問題において，採ると固定した枝集合を F_1，採らないと固定した枝集合を F_0 とする．また，本研究で用いている緩和問題は 2 分割された構造をしており，F_1，F_0 という制限のついた子問題をそれぞれ $L_x(\boldsymbol{\lambda}^L | F_1, F_0)$，$L_y(\boldsymbol{\lambda}^L | F_1, F_0)$ とする．

分枝限定法で留意すべき戦略の 1 つが，分枝変数の選択法である．一般的な線形緩和を用いるときには，分数解となった変数を分枝変数の候補とすることが多いが，本研究の場合，緩和問題が 2 つに分割されており，$L_x(\boldsymbol{\lambda}^L | F_1, F_0)$ は一般に実数解を返し，$L_y(\boldsymbol{\lambda}^L | F_1, F_0)$ は木構造を持つ整数解を必ず返す．

本研究では，$L_y(\boldsymbol{\lambda}^L | F_1, F_0)$ が木構造を持つ解を返すことに着目し，この T 上で F_1 を除く枝のうち，釘付けテストの際に記憶しておいた PegShortage[e] の値の小さいものを優先的に分枝変数として選択する．このルールの

Algorithm BAB(F_1, F_0)

Input: F_1 and F_0 in the current subproblem
Return: Optimal solution. Optimal value is $BestUB$.
solve $LB_x := L_x(\boldsymbol{\lambda}^L|F_1, F_0)$ and $LB_y := L_y(\boldsymbol{\lambda}^L|F_1, F_0)$
if $L_x(\boldsymbol{\lambda}^L|F_1, F_0)$ or $L_y(\boldsymbol{\lambda}^L|F_1, F_0)$ is infeasible **then** return
UB :=CalcUB(T) // T is the solution to $L_y(\boldsymbol{\lambda}^L|F_1, F_0)$.
if $UB < BestUB$
then $BestUB := UB$ and update the current best solution.
if $BestUB < LB_x + LB_y$ **then** return
if ($|F_1| = n - 1$) **then** return // completed under the set F_1
$e = \arg\min\{\text{PegShortage}[e] \mid e \in T \setminus F_1\}$ // next branching
BAB($F_1 \cup \{e\}, F_0$)
BAB($F_1, F_0 \cup \{e\}$)

下で分枝限定法の再帰アルゴリズムは，次の Algorithm BAB(F_1, F_0) のように記述できる．ここで $BestUB$ や $\boldsymbol{\lambda}^L$，n などは大域的領域に宣言されているものとする．また，木になっている枝を固定するので，再帰呼び出しのうち 1 つめ BAB($F_1 \cup \{e\}, F_0$) に入ったときには，LB_y の値は親問題と同じ値を返すため，あらためて解く必要はない．

5. 計算機実験

5.1 Kataoka-Yamada との比較

この節での計算機実験は，先行研究である KY（Kataoka-Yamada [9]）と比較するため，計算環境や問題の生成法はもちろん，乱数シードまでも KY とまったく同一にしている．

アルゴリズムは ANCI C 言語で記述し，ラグランジュ緩和問題の一部である $L_x(\boldsymbol{\lambda})$ を計算するために Gurobi5.0.1 を組み込んでいる．計算機は DELL Precision T7500（Intel Xeon X5680（3.3 GHz）× 2，RAM: 96 GB），OS は RedHat Enterprise Linux v5.5 で Intel C++コンパイラを用いた．

問題は点の数 n，枝の数 m の平面グラフ P_n^m と完全グラフ $K_n^m (m = n(n-1)/2)$ 上において行い，各枝の費用 c_e^s，c_e^r は $[1, R]$ の一様整数乱数で与え，仮定 (1) が成り立つように調整した．一方，所要時間 $t_e^\delta (\delta \in \{s, r\})$ は，費用 c_e^δ に応じて以下の 3 種類の相関を与えるように生成した．

UNCOR：各枝の t_e^δ も独立した $[1, R]$ の一様整数乱数で与える．
WEAK：$t_e^\delta := \lfloor 0.8c_e^\delta + [1, R/5] \rfloor$
STRONG：$t_e^\delta := \lfloor 0.8c_e^\delta + 0.1R + [1, R/100] \rfloor$

いずれの場合も，t_e^δ を発生させた後，仮定 (1) を満足するよう調整し，t_e^s，t_e^r とした．また，予算 B の設定も，KY に合わせて $B := \alpha Rn$ とする．この α を予算パラメータと呼ぶことにする．

実験の視点は，時間/費用の相関（**表 2**），乱数発生の範囲 R（**表 3**），および予算パラメータ α（**表 4**）を変えて行い，10 回の試行の平均値を示す．本研究でのアルゴリズムは，ラグランジュ双対問題，釘付けテスト，分枝限定法の 3 段階で構成されているので，それぞれについて主要な結果と実行時間をみる．最後に総実行時間として，本研究のものと KY の結果とを併記する．

表 1，2，3 において，ラグランジュ双対問題について，#ite はラグランジュ乗数を求める劣勾配法の繰返し回数，gap は上下界値の絶対差である．また#g0 は，10 回の実験のうち，ラグランジュ緩和が終了した時点で gap がゼロ，つまり上下界値が一致して最適解が得られた回数であり，ここでカウントされた例題については，以降の手順である釘付けテストや分枝限定法は行っていない．Lcpu はラグランジュ緩和に要した実行時間（秒）である．

釘付けテストについては，#peg1 が採ると固定された枝数，#peg0 が採らないと固定された枝数である．#peg1 が $n-1$ に一致したときには，解に必要なすべての枝が固定された，つまり最適解が得られたので，続く分枝限定法は行っていない．Pcpu は釘付けテストに要した実行時間（秒）である．

分枝限定法については，opt が最適値であり，個々の例題すべてにおいて KY と一致したことを確認している．#bra は分枝頂点数，BBcpu は分枝限定法に要した計算時間である．ただし，分枝限定法の制限時間を 1,200 秒としているので，一部 1,200 秒以内に解けなかった問題がある．解けなかった問題の数については，各表の下にコメントとして記している．1,200 秒で解けなかった場合は最適値は，1,200 秒が経過した時点での最良値であり，最適性は保証されていない．分枝変数も 1,200 秒経過した時点のものである．また BBcpu は 1,200 秒として 10 回の平均に含めているので，もし最適性が保証されるまで計算を続けると，表の数値よりも大きなものになる．最後の 2 列では本研究での総実行時間（Tcpu）と KY での実行時間（KYcpu）を比較している．

全体を通して，ラグランジュ緩和による上下界値の差は十分小さくなっており，問題の種類によっては上界値が一致することで最適に解けている場合も散見できる（#g0 の数を見よ）．また，ラグランジュ緩和だけでは解けなくても，続く釘付けテストで多くの枝が固定されていることが観察できる．上下界値の差が十分小さく，また多くの枝の釘付けがされているので，分枝限定法に入っても，最適解が得られるまでにわずかな分枝数しか生成されていないことも観察できる．

表 2 は問題の費用と時間との相関の違いを見るものであるが，特に平面的なグラフでは，費用と時間の相関が強くなるほど，ラグランジュ緩和において上下界値が一致する傾向がみられる．さらに，点の数が 1,000 になっても，ラグランジュ双対問題における劣勾配法の繰返し回数は 50 にも至らず，実行時間も 1 秒を切っている．しかしながら，

表 2 費用と時間の相関を変えた場合：$R = 1000$, $\alpha = 0.4$
Table 2 The results for correlation of time and cost: $R = 1000$, $\alpha = 0.4$.

		Lagrangian Dual				Pegging Test			Branch-and-Bound			CPU	
cor.	prob.	#ite	gap	#g0	Lcpu	#peg1	#peg0	Pcpu	opt	#bra	BBcpu	Tcpu	KYcpu
	P^{260}_{100}	272.3	2.69	3	0.06	96.6	158.7	0.06	20140.74	11.4	0.00	0.13	2.23
	P^{560}_{200}	321.8	1.80	2	0.16	195.5	357.6	0.28	37678.31	16.2	0.01	0.45	26.40
U	P^{1120}_{400}	191.9	0.88	2	0.23	395.9	718.4	1.07	76099.86	12.8	0.01	1.31	180.53
N	P^{1680}_{600}	112.4	0.33	1	0.24	596.8	1078.3	2.59	113490.15	9.2	0.01	2.84	752.40
C	P^{2240}_{800}	124.3	0.40	3	0.38	795.3	1435.7	3.57	154109.80	16.4	0.03	3.98	1929.40
O	P^{2800}_{1000}	67.6	0.18	1	0.37	996.4	1799.1	7.16	190702.76	8.0	0.02	7.55	—
R	K^{780}_{40}	350.4	2.10	0	0.18	32.5	736.7	0.61	1136.40	38.8	0.02	0.80	0.39
	K^{3160}_{80}	414.3	1.66	0	0.82	64.1	3068.6	9.18	1283.95	267.8	0.47	10.46	5.84
	K^{7140}_{120}	328.1	1.19	0	1.71	92.5	7008.1	48.79	1339.21	310.0	1.11	51.61	27.89
	K^{12720}_{160}	266.8	0.82	0	2.73	130.3	12546.7	154.86	1373.43	1309.6	9.47	167.05	99.79
	K^{19900}_{200}	353.9	0.63	0	6.79	152.5	19682.3	387.13	1368.59	439.2	4.76	398.68	304.63
	P^{260}_{100}	121.2	0.70	8	0.04	98.0	159.5	0.02	25054.43	0.8	0.00	0.06	1.73
	P^{560}_{200}	46.4	0.16	6	0.15	197.5	359.5	0.15	45487.58	3.0	0.00	0.21	13.89
W	P^{1120}_{400}	35.6	0.02	8	0.15	398.5	720.5	0.28	91612.93	0.1	0.00	0.42	134.22
E	P^{1680}_{600}	50.6	0.09	7	0.31	597.7	1078.0	0.91	137051.04	1.5	0.00	1.22	636.71
A	P^{2240}_{800}	44.8	0.15	7	0.47	796.5	1436.0	1.08	183928.86	1.2	0.00	1.55	1039.86
K	P^{2800}_{1000}	37.7	0.00	9	0.71	999.0	1801.0	0.82	227619.05	0.0	0.00	1.53	—
	K^{780}_{40}	367.0	4.54	1	0.18	31.7	733.8	0.52	2628.55	95.4	0.09	0.78	0.12
	K^{3160}_{80}	457.1	2.63	0	0.86	67.2	3071.2	8.96	3696.54	155.0	0.65	10.47	2.56
	K^{7140}_{120}	414.2	2.09	0	2.14	103.5	7005.2	45.66	4688.30	262.8	2.41	50.21	14.81
	K^{12720}_{160}	416.5	1.64	0	3.46	138.2	12543.1	146.91	5320.90	389.8	17.55	167.91	49.83
	K^{19900}_{200}	339.5	1.51	0	4.85	173.5	19679.5	376.54	6033.00	896.0	60.59	441.98	133.22
	P^{260}_{100}	17.7	0.00	10	0.02	—	—	0.00	28251.38	0.0	0.00	0.02	1.93
	P^{560}_{200}	23.6	0.00	10	0.06	—	—	0.00	51495.75	0.0	0.00	0.06	61.04
S	P^{1120}_{400}	29.4	0.00	10	0.19	—	—	0.00	103681.39	0.0	0.00	0.19	109.47
T	P^{1680}_{600}	35.4	0.00	9	0.36	598.0	1080.0	0.33	155091.17	0.3	0.00	0.69	1734.38
R	P^{2240}_{800}	34.5	0.10	9	0.60	792.0	1430.0	0.56	208458.51	1.7	0.00	1.16	1675.32
O	P^{2800}_{1000}	42.6	0.04	8	0.93	997.5	1796.0	1.74	257683.67	0.3	0.00	2.67	—
N	K^{780}_{40}	414.6	3.05	0	0.20	29.4	732.2	0.60	4659.00	169.4	0.10	0.90	0.11
G	K^{3160}_{80}	338.4	1.09	0	0.66	63.9	3067.7	8.85	8821.20	275.4	0.63	10.13	2.53
	K^{7140}_{120}	304.1	0.79	0	1.44	96.6	7000.7	43.61	12991.80	959.2	5.03	50.07	26.99
	K^{12720}_{160}	143.8	0.48	0	1.60	132.3	12540.9	140.44	17128.10	2879.0	26.24	168.28	191.53
	K^{19900}_{200}	118.5	0.68	0	2.40	148.7	19664.1	383.52	21278.00	15728.2	[1)]426.04	811.96	1703.53

1) 10 問中 3 題が 1,200 秒以内に終了せず，その場合の CPU は 1,200 秒としている．

完全グラフにおいてはラグランジュ緩和だけでは，劣勾配法の繰返し数を多くしても最適解が得られず，釘付けテストをしても多くの枝が未固定のまま残ってしまっている．この状況で分枝限定法に入っても，最適解を得るまでには，ある程度の分枝数や実行時間を要することが分かる．特に強相関の場合には 1,200 秒以内で解けなかった問題もいくつかみられた．

総計算時間に関しては，平面的グラフにおいては本研究で提案するアルゴリズムの方が，KY で提案されているアルゴリズムよりも圧倒的に速く解くことに成功している．一方，完全グラフでは KY に軍配が上がる．しかしながら，いずれの場合も，釘付けテストに要する計算時間が，総計算時間の大半を占めており，枝数の多い完全グラフでは，大きな負担になっていることが分かる．

表 3 は乱数の生成範囲を変えた場合の結果である．乱数の範囲を広げると，おのずと最適値も大きくなるが，その傾向は枝数が限られている平面的グラフの方が顕著である．特に 10^2 から 10^3 に変わる 10 倍よりも，10^3 から 10^4 に変わる 10 倍の影響の方が大きい．これは同じ 10 倍であっても，900（$= 10^3 - 10^2$）に広がるのと，9000（$= 10^4 - 10^3$）に広がるのとの違いもあると考えられる．平面的グラフでは，乱数の範囲が広がると，目的関数値も全体的に大きくなるので，ラグランジュ緩和における gap（絶対誤差）も大きくはなっているが，ラグランジュ双対問題だけで上下界値が一致して最適解が得られる数にはそれほど大きな影響はない．その後の釘付けテストや分枝限定法においても，乱数の範囲を広げることによる影響はほとんどみられない．

表 3 乱数の範囲 R を変えた場合：UNCOR，$\alpha = 0.4$
Table 3 The results for the range of random: UNCOR, $\alpha = 0.4$.

		Lagrangian Dual				Pegging Test			Branch-and-Bound			CPU	
L	prob.	#ite	gap	#g0	Lcpu	#peg1	#peg0	Pcpu	opt	#bra	BBcpu	Tcpu	KYcpu
10^2	P_{100}^{260}	57.6	0.22	3	0.02	96.7	158.3	0.06	2048.66	9.8	0.00	0.08	0.96
	P_{200}^{560}	50.7	0.04	5	0.04	197.4	359.8	0.17	3839.79	2.3	0.00	0.22	6.49
	P_{400}^{1120}	63.4	0.08	2	0.10	396.3	717.5	1.05	7752.10	23.9	0.02	1.18	41.28
	P_{600}^{1680}	60.5	0.04	3	0.18	596.7	1077.9	2.05	11560.10	8.5	0.01	2.24	177.12
	P_{800}^{2240}	55.5	0.16	5	0.25	785.6	1422.4	2.55	15689.80	42.6	0.07	2.87	387.66
	P_{1000}^{2800}	65.1	0.09	2	0.38	991.9	1791.3	6.34	19422.69	45.2	0.10	6.82	—
	K_{40}^{780}	71.0	0.35	0	0.05	31.9	734.3	0.60	130.97	55.2	0.03	0.67	0.19
	K_{80}^{3160}	73.7	0.39	0	0.19	51.9	3057.6	9.00	164.90	4456.2	8.17	17.36	3.74
	K_{120}^{7140}	116.4	0.53	0	0.82	58.5	6956.6	45.88	191.08	13655.2	56.94	103.64	22.94
	K_{160}^{12720}	153.9	0.41	0	2.08	78.3	12479.5	145.51	215.09	52894.8	428.56	576.16	85.49
	K_{200}^{19900}	186.2	0.50	0	4.17	72.6	19528.5	390.94	238.65	55434.6	[2)]851.41	1246.51	2743.46
10^3	P_{100}^{260}	272.3	2.69	3	0.06	96.6	158.7	0.06	20140.74	11.4	0.00	0.13	2.23
	P_{200}^{560}	321.8	1.80	2	0.16	195.5	357.6	0.28	37678.31	16.2	0.01	0.45	26.40
	P_{400}^{1120}	191.9	0.88	2	0.23	395.9	718.4	1.07	76099.86	12.8	0.01	1.31	180.53
	P_{600}^{1680}	112.4	0.33	1	0.24	596.8	1078.3	2.59	113490.15	9.2	0.01	2.84	752.40
	P_{800}^{2240}	124.3	0.40	3	0.38	795.3	1435.7	3.57	154109.80	16.4	0.03	3.98	1929.40
	P_{1000}^{2800}	67.6	0.18	1	0.37	996.4	1799.1	7.16	190702.76	8.0	0.02	7.55	—
	K_{40}^{780}	350.4	2.10	0	0.18	32.5	736.7	0.61	1136.40	38.8	0.02	0.80	0.39
	K_{80}^{3160}	414.3	1.66	0	0.82	64.1	3068.6	9.18	1283.95	267.8	0.47	10.46	5.84
	K_{120}^{7140}	328.1	1.19	0	1.71	92.5	7008.1	48.79	1339.21	310.0	1.11	51.61	27.89
	K_{160}^{12720}	266.8	0.82	0	2.73	130.3	12546.7	154.86	1373.43	1309.6	9.47	167.05	99.79
	K_{200}^{19900}	353.9	0.63	0	6.79	152.5	19682.3	387.13	1368.59	439.2	4.76	398.68	304.63
10^4	P_{100}^{260}	323.8	26.29	4	0.08	96.3	158.8	0.05	201099.23	8.7	0.00	0.13	2.52
	P_{200}^{560}	324.3	17.30	2	0.16	196.1	357.4	0.28	376127.38	15.7	0.01	0.45	28.44
	P_{400}^{1120}	275.7	9.56	3	0.29	395.3	717.0	0.93	759673.91	14.1	0.01	1.24	266.77
	P_{600}^{1680}	277.1	3.97	3	0.45	597.6	1079.4	2.04	1132909.67	5.5	0.01	2.50	1153.13
	P_{800}^{2240}	321.6	2.50	2	0.68	796.4	1437.9	4.07	1538421.63	11.9	0.02	4.77	2989.04
	P_{1000}^{2800}	379.5	2.63	1	1.05	996.7	1798.0	7.21	1903670.67	16.1	0.04	8.29	—
	K_{40}^{780}	500.0	22.02	0	0.24	32.8	736.5	0.60	11189.40	47.6	0.02	0.86	0.47
	K_{80}^{3160}	500.0	13.59	0	0.94	64.8	3071.5	9.15	12482.59	161.8	0.27	10.37	8.64
	K_{120}^{7140}	500.0	10.39	0	2.43	94.5	7009.7	47.18	12842.73	264.6	0.95	50.56	52.11
	K_{160}^{12720}	500.0	7.01	0	5.50	131.8	12547.8	161.86	13011.32	308.4	2.12	169.48	181.27
	K_{200}^{19900}	465.5	4.33	0	8.36	164.6	19688.2	424.93	12783.14	399.2	5.08	438.37	1639.51

2) 10 問中 7 題が 1,200 秒以内に終了せず，その場合の CPU は 1,200 秒としている．

完全グラフにおいては，乱数の範囲が広がると，ラグランジュ緩和による上下界値の差は大きくなるが，その後の釘付けテストや分枝限定法にはそれほど影響は与えていない．しかしながら，乱数の範囲が狭くなると，釘付けテストの結果が顕著に悪くなり，分枝限定法においても 1,200 秒で解けない場合が多数出てきた．これは乱数の範囲が狭くなると，枝の時間や費用に同等のものが増え，最適解の差別化が難しくなったからだと思われる．

表 4 は予算パラメータの変化を見るものである．総予算 B が小さくなると，これまで効率良く解けていた平面的グラフにおいて，急激に解けなくなる場合が出てくる．特にラグランジュ緩和の上下界値の差が目立って悪くなっている．これは，上界値を得る機会が，ラグランジュ双対問題において偶然得られる実行可能解に依存しているからだと考える．予算制約が厳しいと，劣勾配の過程で得られた全域木では，標準計画の費用の合計だけでも予算を超えてしまう確率も高くなり，暫定解に対する情報が激減する．十分良い実行可能解が手元にないままで釘付けテストを行っても，十分な効果が得られないことも表 4 より観察できる．

完全グラフでも同様の傾向はあるものの，費用が十分小さい枝も多くなるので，適当な全域木であっても実行可能性を満足する確率が高い．したがって平面的グラフほど予算パラメータが小さくなったときの影響は目立っていない．

本研究のアルゴリズムによって，特に平面的グラフにおいて KY よりは優れた結果を出したが，完全グラフでは十分な成果が出ていない．この 2 種類のグラフによる実験から，枝密度の違いが影響していることが予想される．予算制約が厳しいときに解き難くなっていることも，枝密度と

表 4 予算パラメータ α を変えた場合：UNCOR，$R = 1000$
Table 4 The results for the budget parameter α: UNCOR, $R = 1000$.

		Lagrangian Dual				Pegging Test			Branch-and-Bound			CPU	
α	prob.	#ite	gap	#g0	Lcpu	#peg1	#peg0	Pcpu	opt	#bra	BBcpu	Tcpu	KYcpu
	P_{100}^{260}	484.2	8.15	0	0.16	93.1	158.2	0.08	39628.31	24.9	0.01	0.25	1.47
	P_{200}^{560}	497.1	8.89	0	0.37	176.8	353.5	0.33	76023.37	217.4	0.09	0.78	22.21
	P_{400}^{1120}	487.1	13.07	0	0.78	343.2	702.3	1.26	153959.10	2162.5	1.84	3.88	183.90
	P_{600}^{1680}	500.0	20.46	0	1.30	485.1	1034.7	2.77	229814.34	23168.8	29.61	33.68	740.04
	P_{800}^{2240}	500.0	20.83	0	1.79	645.8	1374.8	4.95	309795.34	26994.2	46.64	53.38	2784.88
0.2	P_{1000}^{2800}	476.5	23.04	0	2.38	823.5	1717.8	7.77	384965.57	137220.7	[3)]316.62	326.76	—
	K_{40}^{780}	487.8	6.05	0	0.29	29.7	735.6	0.67	3266.57	52.0	0.03	0.99	0.81
	K_{80}^{3160}	464.2	7.24	0	1.16	54.5	3067.4	10.10	3727.89	264.6	0.49	11.75	13.58
	K_{120}^{7140}	497.1	5.49	0	3.02	78.5	7001.6	49.61	4169.93	661.0	2.48	55.11	61.84
	K_{160}^{12720}	500.0	7.82	0	5.87	88.0	12511.7	165.59	4522.27	4102.0	33.79	205.24	199.19
	K_{200}^{19900}	497.1	9.30	0	10.07	99.2	19616.0	410.86	4677.59	9188.4	137.12	558.05	505.21
	P_{100}^{260}	272.3	2.69	3	0.06	96.6	158.7	0.06	20140.74	11.4	0.00	0.13	2.23
	P_{200}^{560}	321.8	1.80	2	0.16	195.5	357.6	0.28	37678.31	16.2	0.01	0.45	26.40
	P_{400}^{1120}	191.9	0.88	2	0.23	395.9	718.4	1.07	76099.86	12.8	0.01	1.31	180.53
	P_{600}^{1680}	112.4	0.33	1	0.24	596.8	1078.3	2.59	113490.15	9.2	0.01	2.84	752.40
	P_{800}^{2240}	124.3	0.40	3	0.38	795.3	1435.7	3.57	154109.80	16.4	0.03	3.98	1929.40
0.4	P_{1000}^{2800}	67.6	0.18	1	0.37	996.4	1799.1	7.16	190702.76	8.0	0.02	7.55	—
	K_{40}^{780}	350.4	2.10	0	0.18	32.5	736.7	0.61	1136.40	38.8	0.02	0.80	0.39
	K_{80}^{3160}	414.3	1.66	0	0.82	64.1	3068.6	9.18	1283.95	267.8	0.47	10.46	5.84
	K_{120}^{7140}	328.1	1.19	0	1.71	92.5	7008.1	48.79	1339.21	310.0	1.11	51.61	27.89
	K_{160}^{12720}	266.8	0.82	0	2.73	130.3	12546.7	154.86	1373.43	1309.6	9.47	167.05	99.79
	K_{200}^{19900}	353.9	0.63	0	6.79	152.5	19682.3	387.13	1368.59	439.2	4.76	398.68	304.63
	P_{100}^{260}	222.3	1.04	4	0.05	96.8	159.3	0.05	12517.86	6.2	0.00	0.10	0.20
	P_{200}^{560}	243.7	0.83	2	0.12	196.5	358.6	0.27	23363.35	9.6	0.01	0.40	1.96
	P_{400}^{1120}	62.3	0.22	3	0.09	396.3	717.71	0.91	46975.47	14.2	0.01	1.01	23.13
	P_{600}^{1680}	70.6	0.57	1	0.17	595.0	1076.9	2.58	69640.77	16.3	0.02	2.77	52.04
	P_{800}^{2240}	62.8	0.06	4	0.22	797.8	1439.5	3.03	94517.33	1.6	0.00	3.25	140.81
0.6	P_{1000}^{2800}	100.2	0.39	5	3.93	992.6	1791.6	3.93	117142.73	15.7	0.03	4.37	—
	K_{40}^{780}	232.9	1.31	2	0.11	32.3	735.9	0.48	646.43	66.3	0.05	0.64	0.07
	K_{80}^{3160}	156.8	0.72	0	0.29	69.8	3073.2	8.91	711.02	141.7	0.34	9.54	2.22
	K_{120}^{7140}	81.0	0.40	0	0.51	108.1	7010.8	46.43	722.35	118.2	0.63	47.57	14.05
	K_{160}^{12720}	91.1	0.50	0	1.00	137.7	12540.4	146.53	735.80	955.8	8.52	156.04	255.68
	K_{200}^{19900}	71.0	0.33	0	1.38	176.6	19679.8	361.95	747.61	821.0	11.67	374.99	584.91

3) 10 問中 2 題が 1,200 秒以内に終了せず，その場合の CPU は 1,200 秒としている．

の関連が深いと考えられる．続く 5.2 節では，さらなる大規模問題において，特に枝密度を変化させた場合と予算制約の厳しい場合について詳しく調べることにする．

5.2 枝密度の影響

前節の実験結果に基づき，本節では枝密度および予算制約を変化させた実験を行う．枝密度の調整は，完全グラフの各枝に対し，その枝を採るか否かをある割合で判定して作成する．たとえば点の数が 200 のグラフで枝密度が 5%としたい場合，完全グラフを想定した 200·199/2=19900 本の各枝に対し，5%の割合で実際に採ることにする．すなわち，平均的に 0.05·19900=995 本の枝を選んで作成する．平均的な枝数なので，ばらつきはあるが，**表 5** では prob. の列に G_{200}^{995} のように表記する．この密度を 5，10，20%について作成し，表 5 では列名を density とした．また，5.1 節の結果で影響が大きかった予算制約についても，特に総予算が少ない場合の影響を見るため，α が 0.2 と 0.4 の場合について実験を行う．点の数についても，枝密度が疎であった平面グラフでは 1000 まで解けていること，完全グラフでは 200 くらいまでしか解けていないことを考慮し，200，400，600，800，1000 について実験する．

各列の意味は，5.1 節の表に用いたものと同じであるが，最後の#slv は，10 問のうち分枝限定法において 1,200 秒で解けた問題数，つまり最適性の保証がある解が得られた問題数である．計算時間に関しては，表 5 の下部の注にあるように，釘付けテストに関しては時間制限を設けず，結果がでるまで実行した．これは，本節では枝密度の違いにより，釘付けテストの効果およびその負担を明らかにする

表 5 枝密度と予算パラメータを変えた場合：UNCOR，$R = 1000$
Table 5 The results for edge-density and budget parameter: UNCOR, $R = 1000$.

den-			Lagrangian Dual			Pegging Test			Branch-and-Bound			Total	
sity	α	prob.	#ite	gap	Lcpu	#peg1	#peg0	Pcpu	opt	#bra	BBcpu	Tcpu	#slv
5	0.2	G^{995}_{200}	425.9	2.39	0.47	188.2	789.9	0.91	51243.90	59.7	0.04	1.42	10
		G^{3990}_{400}	500.0	4.71	2.04	354.8	3558.6	15.45	60669.93	299.6	0.65	18.15	10
		G^{8985}_{600}	477.1	7.12	5.09	496.6	8292.5	76.52	67394.44	20808.5	127.75	209.36	9
		G^{15980}_{800}	488.2	8.59	10.76	635.3	15037.0	251.70	72352.70	22297.3	271.98	534.44	9
		G^{24975}_{1000}	499.4	4.13	19.42	857.6	23856.3	684.67	75985.71	12871.7	252.03	956.12	8
	0.4	G^{995}_{200}	332.9	1.40	0.26	194.8	788.8	0.88	22877.61	21.5	0.01	1.16	10
		G^{3990}_{400}	144.1	0.53	0.50	391.5	3567.1	14.23	23743.57	56.7	0.13	14.85	10
		G^{8985}_{600}	200.7	1.30	1.75	575.1	8306.5	73.67	25034.56	206.6	1.01	76.43	10
		G^{15980}_{800}	129.3	0.77	2.72	769.7	15101.1	242.68	25394.90	393.8	4.03	249.43	10
		G^{24975}_{1000}	171.7	1.07	5.81	944.2	23864.9	664.29	25658.58	633.8	10.52	680.62	10
10	0.2	G^{1990}_{200}	444.2	3.04	0.86	181.3	1790.7	3.93	30044.37	145.5	0.16	4.95	10
		G^{7980}_{400}	481.6	4.20	4.57	345.9	7532.1	62.26	35479.53	490.6	2.03	68.87	10
		G^{17970}_{600}	486.7	3.39	11.67	516.3	17226.9	347.07	39774.48	1268.6	15.16	373.90	10
		G^{31960}_{800}	497.0	5.76	23.52	624.0	30958.1	1208.76	41376.29	10562.3	281.72	1514.01	8
		G^{49950}_{1000}	500.0	24.04	40.47	414.5	48206.6	3254.81	42861.04	23853.0	1048.04	4343.32	2
	0.4	G^{1990}_{200}	167.8	0.75	0.27	194.3	1793.3	3.38	11345.14	32.8	0.04	3.69	10
		G^{7980}_{400}	180.6	0.61	1.54	386.7	7532.0	53.10	12521.47	104.1	0.47	55.11	10
		G^{17970}_{600}	126.7	0.68	2.87	566.8	17240.9	318.90	13031.93	386.2	4.38	326.15	10
		G^{31960}_{800}	194.2	0.61	8.03	736.5	31022.8	1172.38	13366.73	1140.4	26.48	1206.89	10
		G^{49950}_{1000}	253.9	0.69	15.36	916.3	48835.0	2922.58	13431.69	1257.4	44.32	2982.26	0
20	0.2	G^{3980}_{200}	475.9	6.53	1.72	163.3	3771.6	15.97	17192.76	543.4	1.17	18.85	10
		G^{15960}_{400}	497.1	10.40	9.43	272.8	15452.1	273.20	20489.38	15693.8	178.68	461.31	9
		G^{35940}_{600}	495.3	16.46	26.26	326.0	34904.5	1599.42	21945.34	22986.1	736.49	2362.17	4
		G^{63920}_{800}	500.0	35.06	51.95	272.4	61393.5	5398.44	22851.03	18571.9	1084.72	6535.11	1
		G^{99900}_{1000}	500.0	37.49	93.05	163.6	96248.4	13692.41	23781.87	12298.2	1200.00	14985.64	0
	0.4	G^{3980}_{200}	365.5	1.11	0.99	186.8	3774.4	13.97	6116.52	152.4	0.33	15.29	10
		G^{15960}_{400}	197.2	0.50	3.05	373.8	15524.3	245.37	6547.69	311.8	3.00	251.42	10
		G^{35940}_{600}	261.9	0.67	10.87	524.0	35171.5	1542.29	6801.54	1014.8	25.17	1578.33	10
		G^{63920}_{800}	338.2	0.65	26.84	683.2	62944.4	5282.45	6979.04	2272.2	112.62	5421.91	10
		G^{99900}_{1000}	355.6	0.68	46.07	824.3	98670.1	13519.86	7155.97	5674.7	468.35	14034.28	7

釘付けテストの計算時間 Pcpu は，打ち切ることなく結果が出るまで実行した．

ためである．釘付けテストは分枝限定法とは異なり，たとえ多くの時間を要するとしても指数的爆発を見せることはなく，ほぼ安定した時間で確実に終了する．

ラグランジュ双対問題については，問題サイズが大きくなると繰返し回数や計算時間はそれなりに増大するものの，全体に占める負担の割合は小さなものである．また，予算制約が厳しくなると，gap も大きくなることもうかがえる．ただし，$\alpha = 0.4$ くらいだと，問題サイズ大きくなっても，最適値が数万程度であっても，gap が 1 未満の良好な値が得られることも分かった．

ラグランジュ双対問題によって，十分小さな gap は得られても，続く釘付けテストにおいては，枝の数の影響は多大な負担となる．また問題のサイズが大きくなると，時間がかかるだけでなく，釘付けの効果も弱まり，さらに予算制約が $\alpha = 0.2$ のときには，木を構成するのに必要な数の半分にも満たない結果が観察された．そのような場合は，分枝限定法に入っても 1,200 秒では最適解が得られない場合がほとんどである．

この実験の設定では，点の数が 1000 の場合では枝密度が 5%であっても，枝数は平均的に 99,900 本もある．5.1 節の実験で用いた平面的なグラフでの 2,800 本と比べるとかなり多い．点の数が多くなると，枝密度 5%というのは，まだ高密度である．本研究で提案したアルゴリズムは，特に釘付けテストは枝数の影響を大きく受けるが，現実には平面的なグラフを効率的に扱うことができれば，ほとんど問題なく実用的なものと考える．

6. まとめ

本研究では，各枝に標準時間と標準費用が，また特急時間と特急費用が与えられたグラフにおいて，限られた予算制限の中で最小（時間）全域木を求める問題を扱った．この時間と費用のトレードオフ関係は，プロジェクトスケ

ジューリングの分野では古くから研究されてきたものであるが，ネットワーク計画問題においては，単純な最小木問題であっても扱った例は少ない．また，プロジェクトスケジューリングでは，費用の配分が連続的でも簡単に解けるが，最小木問題では NP 困難な問題になる．

先行研究としては，KY（Kataoka-Yamada [9]）があるが，彼らとはまったく別のアプローチからアルゴリズムを提案した．本研究のアプローチが成功した背景には，枝費用を表現するときに，標準費用と特急費用との凸結合で示したことに一因がある．このように定式化すると，ラグランジュ緩和したときに枝費用を与える変数と，枝の採否を決める変数が綺麗に 2 分され，2 つの簡単な問題を解くだけで下界値が得られるようになる．しかも上界値が簡単に得られる仕組みも確立できる．

計算実験により，両上下界の精度は良好であり，KY とまったく同じ設定，同じ計算機環境で比較したところ，平面的なグラフにおいては，1,000 倍以上も高速に解くことができた場合も観察された．これらの成功の要因は，上下界値の差が小さいことと，釘付けテストの効果によるところが大きい．ただし，釘付けテストは枝数が多くなると次第に負担がかかるようになる．しかしながら，現実的には平面的なグラフに準ずる問題が多いと考えられるので，特に枝数の多い高密度なグラフを扱わない限り，効果的なアルゴリズムだといえよう．

参考文献

[1] Aggarwal, V., Aneja, Y.P. and Nair, K.P.K.: Minimal spanning tree subject to a side constraint, *Computers & Operations Research*, Vol.9, pp.287–296 (1982).

[2] Ahuja, R.K., Magnanti, T.L. and Orlin, J.B.: *Network Flows — Theory, Algorithms, and Applications*, Prentice Hall, Englewood Criffs (1993).

[3] Ball, M.O., Magnanti, T.L., Monma, C.L. and Nemhauser, G.L. (Eds.): *Network Models, Handbooks in operations research and management science*, Vol.7, North-Holland (1995).

[4] De, P., Dunne, E.J., Ghoshi, J.B. and Wells, C.E.: Complexity of Discrete Time-Cost Tradeoff Problem for Project Networks, *Operations Research*, Vol.45, pp.302–306 (1997).

[5] Değirmenci, G. and Azizoğlu, M.: Branch and Bound Based Solution Algorithms for the Budget Constrained Discrete Time/Cost Trade-off Problem, *Journal of the Opeerational Research Society*, Vol.64, pp.1474–1484 (2013).

[6] Deineko, V.G. and Woeginger, G.J.: Hardness of Approximation of the Discrete Time/Cost Trade-off Problem, *Operations Research Letters*, Vol.29, pp.207–210 (2001).

[7] Guignard, M. and Rosenwein, M.: An application of Lagrangean decomposition to the resource-constrained minimum weighted arborescence problem, *Networks*, Vol.20, pp.345–359 (1990).

[8] Gurobi Optimizer 5.0, available from ⟨http://www.gurobi.com⟩.

[9] Kataoka, S. and Yamada, T.: Algorithms for the Minimum Spanning Tree Problem with Resource Allocation, *Operations Research Perspectives*, Vol.3, pp.5–13 (2016).

[10] Kellerer, H., Pferschy, U. and Pisinger, D.: *Knapsack Problems*, Springer, Berlin (2004).

[11] Kelley, J.E.: Critical-Path Planning and Scheduling – Mathematical Basis, *Operations Research*, Vol.9, pp.296–320 (1961).

[12] Martello, S. and Toth, P.: *Knapsack Problems – Algorithms and Computer Implementations*, John Wiley & Sons, Chichester (1990).

[13] Li, Y., Zou, C.Y., Zhang, S. and Vai, M.I.: Research on multi-objective minimum spanning tree algorithm based on ant algorithm, *Research Journal of Applied Sciences, Engineering and Technology*, Vol.5, pp.5051–5056 (2013).

[14] 関根智明：PERT/CPM，日科技連 (1973).

[15] 刀根　薫：PERT 講座 I 基礎編，東洋経済新報社 (1966).

[16] Yamada, T. Watanabe, K. and Kataoka, S.: Algorithms to Solve the Knapsack Constrained Maximum Spanning Tree Problem, *International Journal of Computer Mathematics*, Vol.82, pp.23–34 (2005).

[17] Wo, B.Y. and Chao, K.M.: *Spanning Trees and Optimization Problems*, Chapman & Hall/CRC (2004).

片岡 靖詞 （正会員）

防衛大学校情報工学科准教授．1985 年早稲田大学理工学部工業経営学科卒業，1987 年同大学大学院修士課程修了，1990 年同大学院博士課程満期退学．1990 年防衛大学校情報工学科．1993 年博士（工学）．各種の組合せ最適化アルゴリズム開発に従事．日本オペレーションズ・リサーチ学会，INFORMS 各会員．

村崎 暢彦

陸上自衛隊．2010 年 3 月防衛大学校情報工学科卒業，同年陸上自衛隊入隊，幹部候補生学校入校．2011 年 3 月～2013 年 3 月第 11 通信中隊勤務．2013 年 4 月～2015 年 3 月防衛大学校理工学研究科前期課程．2015 年 3 月～9 月技術研究本部勤務．2015 年 10 月～現在防衛装備庁勤務．