頂点容量制約付き有向全域木パッキング問題に対する発見的解法

田中 勇真

Doctoral Dissertation

submitted to Graduate School of Information Science, Nagoya University

in partial fulfillment of the requirement for the degree of

DOCTOR OF INFORMATION SCIENCE

March, 2011

前書き

近年では, アドホックネットワークやセンサーネットワークに対する研究が盛んに行われている. その中に, 端末やセンサーの情報を基地局に収集, もしくは基地局から端末やセンサーに命令を配送する操作が頻繁に行われることが想定されたモデルがある. ただし, 情報を収集するための端末やセンサーにはバッテリーが使用されることが少なくないため, 効率的な通信を実現するための通信トポロジを構築することが求められる. そのような限られたバッテリー容量を用いて, 基地局への収集回数, あるいは基地局からの配送回数を最大化するような通信トポロジを構築する問題はグラフパッキング問題としてモデル化されることが多い.

本研究ではそのような問題を応用の 1 つとして持つ頂点容量制約付き有向全域木パッキング問題を扱う. 頂点容量制約付き有向全域木パッキング問題は, 入力として有向グラフ, 根, 各辺の始点側と終点側の消費量, および頂点容量が与えられ, 根に流入するような有向全域木の集合とそれらの木のパッキング回数を決定する問題である. そのとき, 全ての木をパッキングしたときの各頂点の合計資源消費量が頂点容量を超えないという制約のもとで, 木の合計パッキング回数を最大化することを目的とする.

しかしながら, 頂点容量制約付き有向全域木パッキング問題は強 NP 困難であることが証明されており, 厳密な最適解を現実的な時間で求めることは困難である. そのため, 現実的な時間でなるべく良い解を発見する手法である, 発見的解法を開発することが強く望まれる. 本研究では, 本問題に対して 2 つの発見的解法を提案する. 2 つの発見的解法では, 本問題に対する緩和問題を定義し, 緩和問題を解いたときの情報を用いて本問題に対する実行可能解を生成する. 緩和問題とは, 元の問題の制約条件を緩めたり, 取り除いたりした問題であり, 元の問題に比べて解きやすい問題となる. 元の問題を解くことが困難である場合によく用いられる手法であり, 本研究で提案するアルゴリズムも緩和問題から得られる情報を用いて本問題を解く. 具体的な

緩和法としては, 線形緩和とラグランジュ緩和を用いた.

さらに, 本研究では提案したアルゴリズムの性能を評価するために計算実験を行った. 計算実験の際には, 提案手法の性能が既存アルゴリズムに比べて相対的にどの程度優れているのかを判断するために比較実験も行った. そのような実験を通して, 提案した 2 つのアルゴリズムは現実的な時間で既存アルゴリズムよりも良い解を生成できることを観測した. また, ラグランジュ緩和に基づくアルゴリズムの方が線形緩和問題に基づくアルゴリズムよりもさらに実用的であることを示した.

頂点容量制約付き有向全域木パッキング問題に関するいくつかの理論的な結果も得られた. 具体的には, 整数計画問題による多項式サイズの定式化, 線形緩和問題に対する価格付け問題が多項式時間で解けること, および線形緩和に基づくアルゴリズムの近似精度などについて示した.

実際の応用においては, 様々な付加的な制約が必要であることが少なくなく, 本研究で与えた 2 つのアルゴリズムが直接適用できない問題が多数存在する. しかし, 似通った構造を持つ問題であれば, それぞれの問題に特化したアルゴリズムを設計する上での指針や, 内部で呼び出す部品として本研究の成果を活用することができる. また, 本研究で与えた理論的な結果は, 頂点容量制約付き有向全域木パッキング問題に対する興味深い知見を与えてくれる. これらの成果が, センサーネットワークに関する分野, グラフパッキングに関する分野, さらには最適化分野の発展に貢献できれば幸いである.

2011 年 3 月

田中 勇真

目次

図目次

表目次

アルゴリズム目次

第1章　序論

本研究では頂点容量制約付き有向全域木パッキング問題を扱う．頂点容量制約付き有向全域木パッキング問題は, 入力として有向グラフ, 根, 各辺の始点側と終点側の消費量, および頂点容量が与えられ, 根に流入するような有向全域木の集合とそれらの木のパッキング回数を決定する問題である．このとき, 全ての木をパッキングしたときの各頂点の合計資源消費量が頂点容量を超えないという制約のもとで, 木の合計パッキング回数を最大化することを目的とする．この問題は強 NP 困難であることが証明されており, 優れた発見的解法を開発することが求められる．

本研究では, 頂点容量制約付き有向全域木パッキング問題に対して 2 つの発見的解法を提案する．2 つの発見的解法では, 本問題に対する緩和問題に基づいており, 緩和問題を解いたときの情報を用いて本問題に対する実行可能解を生成する．緩和問題とは, 元の問題の制約条件を緩めたり, 取り除いたりすることで, 元の問題に比べて解き易くした問題である．元の問題を解くことが困難である場合によく用いられる手法であり, 本研究で提案するアルゴリズムも緩和問題から得られる情報を用いて本問題を解く．提案するアルゴリズムの 1 つ目は線形緩和に基づいたものであり, 線形緩和問題に対して成り立つ良い性質を用いてアルゴリズムを提案する．2 つ目はラグランジュ緩和に基づいたものであり, ラグランジュ緩和問題に対する優れた発見的解法である劣勾配法を用いたアルゴリズムを提案する．また, 提案する 2 つのアルゴリズムは実用的な問題例に対して, 既存研究より速く, より良い解を生成できることを示す.

各章で述べる内容は次の通りである．本章では, 頂点容量制約付き有向全域木パッキング問題を理解する上で必要な知識と定義を与える．本章の 1.1 節で本問題の背景, 1.2 節で頂点容量制約付き有向全域木パッキング問題の厳密な定義, 1.3 節で本問題の計算複雑性についての議論, 1.4 節で本研究が対象とする問題例の性質, 1.5 節で問題の定式化を示す．続く第 2 章では, 頂点容量制約付き有向全域木パッキング問題

に対して線形緩和に基づくアルゴリズムを提案し, 精度の良い解を生成できることを示す. 第3章では, 第2章で提案したアルゴリズムを含むような一般化された線形緩和に基づくアルゴリズムを定義し, そのようなアルゴリズムの近似精度について議論する. 第4章では, 頂点容量制約付き有向全域木パッキング問題に対してラグランジュ緩和に基づくアルゴリズムを提案し, 第2章で提案したアルゴリズムよりも実用的に優れていることを示す. 最後に第5章で本研究の結論を述べる.

1.1 背景

頂点容量制約付き有向全域木パッキング問題の重要な応用の1つとしてセンサーネットワークがある. 近年では, アドホックネットワークやセンサーネットワークに対する研究が盛んに行われている. 情報を収集するための端末やセンサーにはバッテリーが使用されることが少なくない. そのため, 効率的な通信を実現するための通信トポロジを構築することが求められる. 限られたバッテリー容量を用いてそのような通信回数を最大化するような通信トポロジを構築する問題は総称してネットワークライフタイム問題と呼ばれる. ネットワークライフタイム問題の中にはグラフパッキング問題としてモデル化されているものが多数あり, その中の重要な問題として頂点容量制約付き全域部分グラフパッキング問題がある [5, 17, 26]. 例えば, センサーネットワークでは, ある1つの全域部分グラフは, 全てのセンサーの情報を基地局に伝える情報経路, もしくは基地局の情報を全センサーに伝える情報経路1つに相当する. センサー間, およびセンサー基地局間でデータを送受信すると電力を消費し, 送信側では2点間の距離に依存した電力を必要とする. センサーは通常バッテリーを使用するので, 電力消費量を抑えた情報経路を選択し, なるべく長くセンサーが使えることを考慮しなければならない. この問題に対しては情報経路として高さを制限した全域有向木を用いたLEACH-Cと呼ばれるアルゴリズムが提案されている [17]. さらに効率的な送信経路の選択方法として, 整数計画問題に定式化した後に, 近似解法を用いるものも存在する [26]. Calinescuら [5] のモデルでは, 流入消費量は考えられておらず, 各頂点の消費量は流出消費量の最大値で与えられており, 全域部分グラフに対する条件に応じて様々な問題を考慮した上で, それらの困難性や近似解法が示されている.

頂点容量制約付き有向全域木パッキング問題はグラフに有向全域木をパッキングする問題である. 与えれたグラフに特定の性質を満たす部分グラフをパッキングする問題, つまり辺素な部分グラフの集合を見つける問題は, グラフ理論および離散最適化の基本的な問題として広く研究されている. 例えば, グラフに閉路 [6, 21], 道 [24], 木 [15] などをパッキングする問題がある. とくに, このようなグラフ上に部分グラフをパッキングする問題の中で, 有向グラフに有向全域木をパッキングする問題は本研究で扱う問題と強い関連があり, 本研究でも理論的な性質を証明するためにそれらの結果を用いている. ここでいう有向全域木とは全ての辺が根と呼ばれる特別な頂点に流入する方向に向き付されている根付き全域木のことである (根から流出する方向で定義されることも多いが, 等価な問題である). この問題は辺素有向全域木パッキング問題と呼ばれ, (多重辺が許される) 有向グラフ $G = (V, E)$ と根 r が与えられ, 同じ辺を使用ない r に流入するような有向全域木の最大パッキング回数を見つける問題と定義される. この問題に対しては, 1973 年に Edmonds によって証明された最大最小定理 [9] を皮切りに, 多くの拡張や効率的なアルゴリズムが提案されている. Tarjan [28] と Lovász [22] は $O(k^2|E|^2)$ 時間のアルゴリズムを, Tong と Lawler [29] は $O(k|V||E| + k^3|V|^2)$ 時間アルゴリズムを, Gabow [12] は $O(k^2|V|^2 + |E|)$ 時間アルゴリズムを提案した. 最近では Bhalgat ら [3] が $O(k|E| \log |V| + k^4|V| \log^2 |V|)$ 時間アルゴリズムを提案している. ここで, k は辺素な有向全域木の数を意味する. さらに, 近年, Kamiyama ら [20] は Edmonds の定理を拡張できることを示した. Fujishige [11] はこの結果を用いて更なる一般化を与えている. また, 各辺に容量がついた辺容量制約付きの辺素有向全域木パッキング問題に対しても研究がなされており, いくつかの多項式時間アルゴリズムが与えられている [14, 23, 25]. とくに, Gabow ら [14] は $O(\min\{|V|, \log C\}|V|^2|E| \log(|V|^2/|E|))$ 時間アルゴリズムを提案している (C は最大の辺容量を表す).

1.2 問題

この節では頂点容量制約付き有向全域木パッキング問題の定義を与える. 頂点容量制約付き有向全域木パッキング問題は, 入力として有向グラフ $G = (V, E)$, 根 $r \in V$, 各有向辺における始点側と終点側の消費量 $t : E \to \mathbb{R}_+$ と $h : E \to \mathbb{R}_+$ ($\mathbb{R}_+$ は非負

実数の集合を表す), 各頂点 $i \in V$ の容量 $b_i \in \mathbb{R}_+$ が与えられる. 便宜上, グラフ G 上の各頂点 $i \in V \setminus \{r\}$ から根 r への有向道が存在するような木 (根 r に流入するような全ての有向全域木) の集合を T_{all} と定義する. また, $\delta_j^+(i)$ と $\delta_j^-(i)$ をそれぞれ木 $j \in T_{\text{all}}$ における頂点 $i \in V$ から出る辺の集合と入る辺の集合とする. 木 $j \in T_{\text{all}}$ の頂点 $i \in V$ における資源消費量を次のように定義する:

$$a_{ij} = \sum_{e \in \delta_j^+(i)} t(e) + \sum_{e \in \delta_j^-(i)} h(e). \tag{1.1}$$

すなわち, 資源消費量 a_{ij} は, 頂点 i から出る木 j の終点側の消費量と, 頂点 i に入る木 j の全ての辺の始点側の消費量の合計である. 頂点容量制約付き有向全域木パッキング問題は, 頂点容量制約

$$\sum_{j \in T} a_{ij} x_j \leq b_i, \quad \forall i \in V \tag{1.2}$$

を満たす有向全域木の部分木集合 $T \subseteq T_{\text{all}}$ と, その部分木集合の各木 $j \in T$ に対してパッキング回数 $x_j \in \mathbb{Z}_+$($\mathbb{Z}_+$ は非負整数の集合を表す) を決定する問題であり, 全ての木のパッキング回数の合計 $\sum_{j \in T} x_j$ を最大化するのが目的である. また, とくに断らない限り, “木” は根 r に流入するような有向全域木のことを意味する.

頂点容量制約付き有向全域木パッキング問題に対する具体例 (問題例) を図 1.1 に示す. 図 1.1 の問題例は 4 つの頂点を持ち, r が根, 1 から 3 までが通常の頂点である. 辺上の 2 つの数字がそれぞれ始点側と終点側の消費量を, 各頂点の側にある影付きの数字が容量を表す. この問題例のグラフとパラメータは次の通りである:

$$\begin{aligned}
&V = \{r, 1, 2, 3\},\\
&E = \{(1, r), (2, r), (1, 2), (2, 1), (3, 1), (3, 2)\},\\
&b(1) = 14,\ b(2) = 13,\ b(3) = 7,\ b(r) = +\infty,\\
&t((1, r)) = 4,\ t((2, r)) = 3,\ t((1, 2)) = 1,\\
&t((2, 1)) = 1,\ t((3, 1)) = 2,\ t((3, 2)) = 4,\\
&h((1, r)) = 0,\ h((2, r)) = 0,\ h((1, 2)) = 3,\\
&h((2, 1)) = 1,\ h((3, 1)) = 2,\ h((3, 2)) = 1.
\end{aligned}$$

図 1.2 は図 1.1 のグラフ上に存在する全ての木 (すなわち, T_{all} の全要素) を列挙したものである. 各頂点上の数字は式 (1.1) による頂点の消費量を表す. 例えば, 木 (a) に

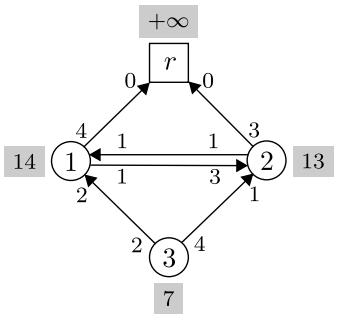


図 1.1: 問題例

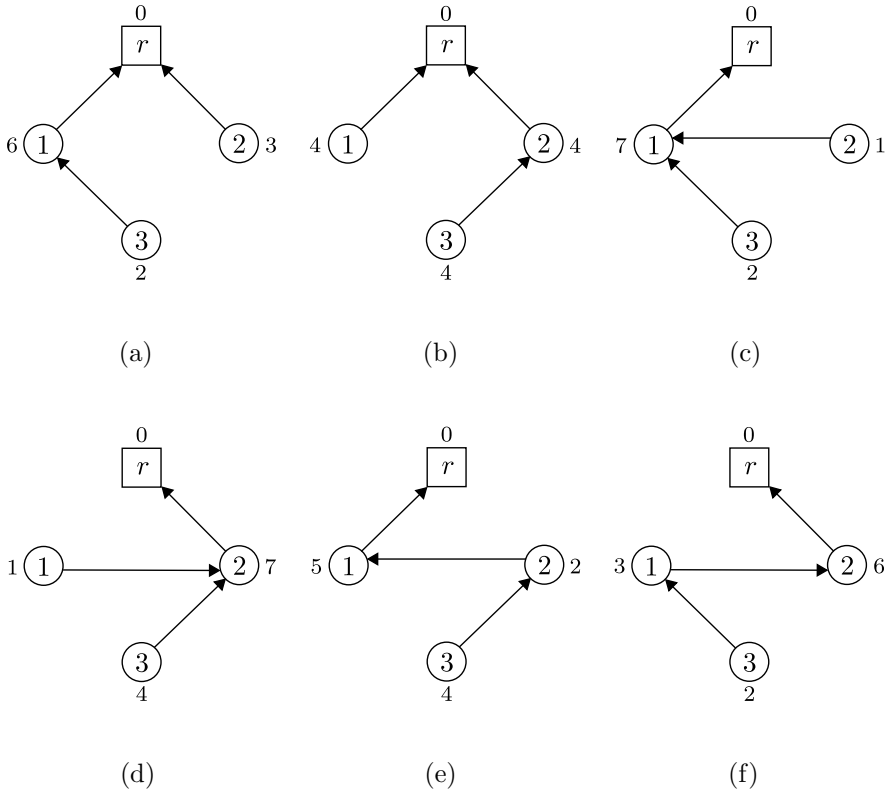


図 1.2: 図 1.1 のグラフ上に存在する全ての木 (T_{all} の全要素)

おける頂点 “1” の資源消費量は $a_{1\mathrm{a}} = t((1,r)) + h((3,1)) = 4+2 = 6$ となる. もし木 (c) を1本, 木 (f) を2本パッキングしたとすると (すなわち, 部分木集合を $T = \{\mathrm{c}, \mathrm{f}\}$ とし, 2つの木のパッキング回数を $x_{\mathrm{c}} = 1$ および $x_{\mathrm{f}} = 2$ としたとすると), この解は頂点容量制約 (1.2) を全ての頂点において満たすので実行可能である. 例えば, 頂点 “1” は $\sum_{j\in T} a_{1j}x_j = 7\times 1 + 3\times 2 = 13 \leq b_1 = 14$ を満たす. また, この解は最適値 $\sum_{j\in T} x_j = 3$ を達成する解である.

1.3 計算複雑性

1.2節で定義した頂点容量制約付き有向全域木パッキング問題は Imahori ら [19] により強 NP 困難であることが証明されている. 彼らは様々な制限を加えた問題に対して計算複雑性を検証しており, 各辺の消費量がユークリッド距離の関数で定義されるような実用的な問題例であっても, 一般的には強 NP 困難であることを示している. 彼らの示した結果は次の通りである:

- 頂点容量制約 (1.2) を満たすような木がグラフ G 上に存在するか否かを判定する問題
  - 始点側の消費量 t だけが存在する ($\forall e \in E, h(e) = 0$) 問題: 多項式時間アルゴリズムが存在
  - 終点側の消費量 h だけが存在する ($\forall e \in E, t(e) = 0$) 問題: 強 NP 困難
    - 全ての辺に対する終点側の消費量が同一である問題: 強 NP 困難
    - 無閉路グラフ: 強 NP 困難
    - 各辺の終点側の消費量が端点間の L^p 距離の関数で定義される問題: 強 NP 困難
- 頂点容量制約付き有向全域木パッキング問題
  - 始点側の消費量 t だけが存在する ($\forall e \in E, h(e) = 0$) 問題: 強 NP 困難
    - 無閉路グラフ: 多項式時間アルゴリズムが存在
    - 各辺の始点側の消費量が端点間の L^p 距離の関数で定義される問題: 強 NP 困難

- 始点側と終点側の消費量 t と h が両方存在する問題: 強 NP 困難
  - 全ての辺の消費量が同一である無閉路グラフ上の問題: 多項式時間アルゴリズムが存在

ここで, 頂点 v の d 次元の座標を $(x_1(v), \ldots, x_d(v)) \in \mathbb{R}^d$ とし, p を 1 以上の実数または $p = \infty$ である定数とすると, 頂点 v と w の間の L^p 距離は $p \neq \infty$ のとき $L^p(v,w) = (|x_1(v) - x_1(w)|^p + \cdots + |x_d(v) - x_d(w)|^p)^{1/p}$, $p = \infty$ のとき $L^\infty(v,w) = \max\{(|x_1(v) - x_1(w)|, \ldots, |x_d(v) - x_d(w)|\}$ により定義される. とくに, L^2 距離はユークリッド距離と呼ばれる.

ここでは, 本研究で取り扱う問題が強 NP 困難であることを示す結果の一例として, 終点側の消費量 h だけが存在し, 無閉路グラフであるような問題例に対して, 頂点容量制約 (1.2) を満たすような木がグラフ G 上に存在するか否かを判定する問題が強 NP 困難であることの証明を紹介する.

定理 1 (Imahori ら [19])**.** 頂点容量制約 (1.2) を満たすような木がグラフ G 上に存在するか否かを判定する問題は強 NP 困難である. また, 与えられたグラフが無閉路グラフであり, しかも始点側の消費量 t が常に 0 である場合に限定しても強 NP 困難のままである.

証明. ビンパッキング問題が上記の判定問題に多項式時間帰着可能であることを示す. ビンパッキン問題は強 NP 完全であることが知られている [16]. B をビンの集合とし, $c_B \in \mathbb{Z}_+$ をビンの容量とする. I をビンに詰め込まれるアイテムの集合とし, 各アイテム $i \in I$ のサイズを $s_i \in \mathbb{Z}_+$ とする. $\max_{i \in I}\{s_i\}$ と c_B がアイテムの要素数 $|I|$ に関する多項式で抑えられる場合に限定しても, ビンパッキング問題は NP 完全のままであることに注意する必要がある. ビン集合 B に対して頂点集合 $U := \{u_1, \ldots, u_{|B|}\}$, アイテム集合 I に対して頂点集合 $W := \{w_1, \ldots, w_{|I|}\}$ を定義する. また, r を根とする. 頂点集合 W の各頂点 $w \in W$ から頂点集合 U の全ての頂点 $u \in U$ に対して, および頂点集合 U の各点 $u \in U$ から根 r に対して辺を張る. 頂点集合 W の頂点 $w \in W$ から出る辺 e の終点側の消費量を $h(e) := s_i$, それ以外の辺 e の終点側の消費量を $h(e) := 0$ とする. 頂点集合 U の各頂点 $u \in U$ の容量を $b_u := c_B$, それ以外の頂点 i の容量を $b_i := 0$ とする. この変換の例を図 1.3 に示す. このようにして得られた問題例のサイズはアイテムの要素数 $|I|$ に関する多項式で抑えられ, 問題例の中に現れ

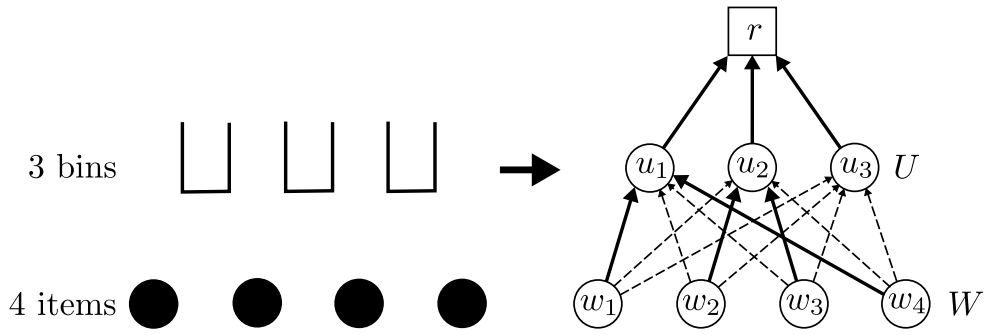


図 1.3: ビンパッキング問題から木の存在判定問題への変換例

る最大の数値は元のビンパッキング問題に対する問題例と同じである. 明らかに, 頂点容量制約 (1.2) を満たすような木が存在することと, 実行可能なビンへのアイテムの詰め込みが存在することは等価である. □

1.4 対象とする問題例

1.3 節で述べたように頂点容量制約 (1.2) を満たすような木を見つける問題は強 NP 困難である. しかしながら, 本研究では頂点容量制約 (1.2) を満たすような木を見つけるのが困難であるような問題例は対象としない. すなわち, グラフ上に少なくとも 1 回はパッキングできる木が多く存在するような問題例を対象とする. 例えば, $\delta^+(i)$ および $\delta^-(i)$ をそれぞれ頂点 $i \in V$ における出る辺と入る辺の集合と定義すると, 全ての頂点 $i \in V$ において

$$\max_{e \in \delta^+(i)} t(e) + \sum_{e \in \delta^-(i)} h(e) \leq b_i \tag{1.3}$$

が満たされるような問題例では, 任意の木が少なくとも 1 回はパッキングできる. これは直感的には始点側と終点側の消費量が頂点容量よりも十分に小さい場合を意味している. この式 (1.3) の仮定は厳しいものではなく, センサーネットワークなどのネットワークに関する実用的な問題例にはそのような仮定を満たす (通信路を見つけることが簡単である) ものが少なくない. また, 頂点容量制約 (1.2) を満たすような木を見つける問題と各木のパッキング回数を最大化する問題は問題の構造が大きく異なり, 別々のアルゴリズムを設計したほうが良いアルゴリズムを開発できる. さ

らに, 式 (1.3) の仮定を満たす問題例に限定しても本問題は強 NP 困難のままである. なぜなら, 文献 [19] 中の Theorem 7 で強 NP 困難を証明するために使用した問題例が式 (1.3) を満たすからである.

1.5 定式化

第 2 章と第 4 章で述べる提案アルゴリズムの準備として, 1.2 節で定義した頂点容量制約付き有向全域木パッキング問題の定式化を行う. 頂点容量制約付き有向全域木パッキング問題は以下のような整数計画問題に定式化できる:

$$\begin{aligned} &\text{maximize} && \sum_{j \in T_{\text{all}}} x_j, \\ &\text{subject to} && \sum_{j \in T_{\text{all}}} a_{ij} x_j \leq b_i, \quad \forall i \in V, \\ & && x_j \geq 0, \qquad\qquad x_j \in \mathbb{Z}, \ \forall j \in T_{\text{all}}. \end{aligned} \tag{1.4}$$

各記号の意味は以下の通りである:

- V: 頂点集合,
- T_{all}: 根 $r \in V$ に流入するような全ての有向全域木の集合,
- a_{ij}: (式 (1.1) で定義される) 木 $j \in T_{\text{all}}$ における頂点 $i \in V$ の資源消費量,
- b_i: 頂点 $i \in V$ の容量,
- x_j: 木 $j \in T_{\text{all}}$ のパッキング回数,
- $\mathbb{Z}$: 整数集合.

1.2 節および式 (1.4) で, T_{all} を根 $r \in V$ に流入するような全ての有向全域木の集合と定義した. しかしながら, T_{all} の要素数は一般的には指数的に大きくなり, T_{all} を直接扱うことは困難である. そこで, 提案するアルゴリズムの内部では部分木集合 $T \subseteq T_{\text{all}}$ に対する次のような問題を計算の対象とする:

$$\begin{aligned} P(T) \quad &\text{maximize} && \sum_{j \in T} x_j, \\ &\text{subject to} && \sum_{j \in T} a_{ij} x_j \leq b_i, \quad \forall i \in V, \\ & && x_j \geq 0, \qquad\qquad x_j \in \mathbb{Z}, \ \forall j \in T. \end{aligned}$$

もし $T = T_{\text{all}}$ のときは, 問題 $P(T_{\text{all}})$ は元の問題 (1.4) と等価である. 任意の木集合 $T \subseteq T_{\text{all}}$ に対して, $P(T)$ の任意の実行可能解は $P(T_{\text{all}})$ の実行可能解になることに注意する必要がある. また, この問題 $P(T)$ の最適値を $\mathrm{OPT}_{P(T)}$ とする. 具体的にどのような部分木集合 $T \subseteq T_{\text{all}}$ を扱うかについては第 2 章と第 4 章で述べる. しかしながら, たとえ部分木集合 T が十分小さなサイズであったとしても $P(T)$ は整数計画問題のままである. 一般的に整数計画問題を厳密に解くことは難しい. そこで, 第 2 章と第 4 章で提案するアルゴリズムでは, $P(T)$ の緩和問題を考え, 緩和問題を解くことによって得られる情報を利用して $P(T)$ の実行可能解を探索する.

式 (1.4) では全ての木の集合 T_{all} を用いて定式化しており, 変数の数が指数的に大きくなってしまう. そのため, 整数計画ソルバーに直接入力することができない. しかしながら, 本研究では, 頂点容量制約付き有向全域木パッキング問題に対して次のような式 (1.4) とは別の定式化が存在することを示した:

$$\text{maximize} \quad f \tag{1.5}$$

$$\text{subject to} \quad \sum_{w:(v,w)\in E} y_{vwi} - \sum_{w:(w,v)\in E} y_{wvi} = \begin{cases} 0 & (v \in V \setminus \{i, r\}) \\ f & (v = i) \\ -f & (v = r) \end{cases} \qquad (\forall i \in V \setminus \{r\}) \tag{1.6}$$

$$y_{vwi} \leq z_{vw} \quad (\forall (v, w) \in E, \forall i \in V \setminus \{r\}) \tag{1.7}$$

$$\sum_{w:(v,w)\in E} z_{vw} = f \quad (\forall v \in V \setminus \{r\}) \tag{1.8}$$

$$\sum_{w:(v,w)\in E} t((v, w))z_{vw} + \sum_{w:(w,v)\in E} h((w, v))z_{wv} \leq b_v \quad (\forall v \in V) \tag{1.9}$$

$$y_{vwi} \geq 0,\ y_{vwi} \in \mathbb{Z} \quad (\forall (v, w) \in E, \forall i \in V \setminus \{r\}) \tag{1.10}$$

$$z_{vw} \geq 0,\ z_{vw} \in \mathbb{Z} \quad (\forall (v, w) \in E) \tag{1.11}$$

$$f \geq 0,\ f \in \mathbb{Z} \tag{1.12}$$

各記号の意味は以下の通りである:

V: 頂点集合,

E: 辺集合,

r: 根,

$t((v,w))$: 辺 $(v,w) \in E$ の始点側の消費量,

$h((v,w))$: 辺 $(v,w) \in E$ の終点側の消費量,

b_v: 頂点 $v \in V$ の容量,

y_{vwi}: 頂点 $i \in V \setminus \{r\}$ から根 r へ f 単位のフローを流したときに辺 $(v,w) \in E$ に流れるフロー量,

z_{vw}: 辺 $(v,w) \in E$ を使用した (グラフ上にパッキングした) 回数,

f: 各頂点 $i \in V \setminus \{r\}$ から根 r へ流すフロー量,

$\mathbb{Z}$: 整数集合.

各制約式の意味を簡単に説明する. 制約式 (1.6) は各頂点 $i \in V \setminus \{r\}$ から根 r へ f 単位のフローを流したときのフロー保存則を表す. 制約式 (1.7) は, 各頂点 $i \in V \setminus \{r\}$ から根 r へフローを流したときに各辺 $(v,w) \in E$ に流れるフロー量が, 辺の使用回数以下でなければならないことを表す. 制約式 (1.8) は全ての頂点 $i \in V \setminus \{r\}$ から出る辺の使用回数の合計が f でなければならないことを表す. 制約式 (1.9) は式 (1.2) で示した頂点容量制約を表し, 頂点容量を超えないように全ての辺 $(v,w) \in E$ の使用回数 (グラフ上にパッキングした回数) を定めなければならないことを意味する. また, 式 (1.10), 式 (1.11) および式 (1.12) は各変数が整数であることを表す. なお, 制約式 (1.8) を取り除いたとしても最適値が変わらないことを示すことができる. したがって, 式 (1.8) は制約として冗長であるが, 解空間を限定することで整数計画ソルバーの探索を効率化する目的のために導入した.

ここからは, 式 (1.4) による定式化と式 (1.5),...,(1.12) による定式化の等価性について議論する. もし, この定式化の実行可能解が与えられれば, z_{vw} より頂点容量制約を満たすような各辺 $(v,w) \in E$ の使用回数を知ることができる. しかし, 各辺の使用回数からいくつかの木のパッキングとして構成できるかどうかは自明ではない. すなわち, E_j を木 $j \in T$ の辺集合とすると, 全ての辺 $(v,w) \in E$ に対して $\sum_{j \in T} |\{(v,w)\} \cap E_j| x_j \leq z_{vw}$ を満たし, しかも $\sum_{j \in T} x_j = f$ となるような木集合 T と各木 $j \in T$ のパッキング回数 x_j が存在するかどうかは分からない. そこで, 次のような問題を考える. 入力として有向グラフ $G = (V, E)$, 根 $r \in V$, 各辺 $(v,w) \in E$ に容量 $z_{vw} \in \mathbb{Z}_+$ が与えられる. 各辺 $(v,w) \in E$ の合計使用回数 $\sum_{j \in T} |\{(v,w)\} \cap E_j| x_j$ が容量 z_{vw} 以下であるような部分木集合 $T \subseteq T_{\text{all}}$ と, 各木 $j \in T$ のパッキング回数

$x_j \in \mathbb{Z}_+$ を決定し, 全ての木のパッキング回数の合計 $\sum_{j\in T} x_j$ を最大化することを目的とする問題である.

この問題は, 1.1 節で述べた辺容量制約付きの辺素有向全域木パッキング問題である. この問題に対しては Gabow ら [14] によって, 高々多項式種類の相異なる木を使用した最適解を出力する多項式時間アルゴリズムが存在することが示されている. 具体的には, 高々$|V| + |E| - 2$ 本の相異なる木を使用した最適解を出力する $O(\min\{|V|, \log z_{\max}\}|V|^2|E| \log(V|^2/|E|))$ 時間のアルゴリズムが存在する. ここで, $z_{\max} = \max_{(v,w)\in E}\{z_{vw}\}$ である. すなわち, 式 (1.5),...,(1.12) による定式化の実行可能解が与えられれば, 全ての辺 $(v, w) \in E$ に対して $\sum_{j\in T} |\{(v, w)\} \cap E_j| x_j \leq z_{vw}$ を満たすような多項式サイズの木集合 T と各木 $j \in T$ のパッキング回数 x_j を多項式時間で見つけることができ, 式 (1.4) による定式化の実行可能解となる.

次に, このようにして得られた式 (1.4) による定式化の実行可能解の目的関数値が $\sum_{j\in T} x_j = f$ を満たすことを示す. そのために, 辺素有向全域木パッキング問題に対する基本的な定理である Edmonds の最大最小定理 [9] を使用する.

定理 2 (Edmonds の最大最小定理 [9])**.** (多重辺が許される) 有向グラフ $G = (V, E)$ と根 $r \in V$ が与えられたとき, r を根とする f 本の辺素な有向全域木が存在する必要十分条件は, 全ての頂点 $k \in V \setminus \{r\}$ に対して k から r への f 本の辺素な有向道が存在することである.

この定理は辺容量制約付きではない辺素有向全域木パッキング問題に対するものであるが, 各辺 $(v, w) \in E$ の容量 z_{vw} は頂点 v と w 間の多重辺の本数と見なすことできるので, 辺容量制約付きの辺素有向全域木パッキング問題に拡張できることがすぐに分かる. すなわち, 次の系が直ちに得られる:

系 1. 有向グラフ $G = (V, E)$, 根 $r \in V$ と各辺 $(v, w) \in E$ に容量 $z_{vw} \in \mathbb{Z}_+$ が与えられたとき, r を根とする f 本の有向全域木がグラフ G 上にパッキングできる必要十分条件は, 全ての頂点 $k \in V \setminus \{r\}$ に対して k から r へ f 単位の (各辺 $(v, w) \in E$ に流れるフローの単位が整数値に限るような) フローが存在することである.

式 (1.5),...,(1.12) による定式化の実行可能解中の z_{vw} より定義された, 上記の辺容量制約付きの辺素有向全域木パッキング問題は, 明らかに全ての頂点 $k \in V \setminus \{r\}$ に対して k から r へ f 単位のフローが存在する. よって, 系 1 より Gabow ら [14] の

アルゴリズムによって得られる式 (1.4) による定式化の実行可能解の目的関数値は $\sum_{j\in T} x_j = f$ を満たす.

以上の議論より, 次の定理が成り立つ:

定理 3. 頂点容量制約付き有向全域木パッキング問題に対して, 式 (1.4) による定式化に目的関数値が f である実行可能解が存在する必要十分条件は, 式 (1.5),...,(1.12) による定式化に目的関数値が f である実行可能解が存在することである.

証明. 式 (1.5),...,(1.12) による定式化に目的関数値が f である実行可能解が存在するとき, 式 (1.4) による定式化に目的関数値が f である実行可能解が存在することは, 上記のように Gabow ら [14] のアルゴリズムを使用することで証明できる. 式 (1.4) による定式化に目的関数値が f である実行可能解が存在すると仮定する. この実行可能解の木集合を T, 各木 $j \in T$ のパッキング回数を x_j とする. 全ての辺 $(v, w) \in E$ に対して $z_{vw} := \sum_{j\in T} |\{(v, w)\} \cap E_j| x_j$ とすると, 明らかに, 各辺 $(v, w) \in E$ の辺容量 z_{vw} を超えないという条件を満たしつつ, 各頂点 $k \in V \setminus \{r\}$ に対して k から r へ f 単位のフローを流すことができる. すなわち, 式 (1.6) と (1.7) を満たすような, 全ての頂点 $i \in V \setminus \{r\}$ と全ての辺 $(v, w) \in E$ に対する y_{vwi} を定めることができる. また, このように定めた z_{vw} は明らかに式 (1.9) を満たす. よって, 式 (1.5),...,(1.12) による定式化に目的関数値が f であるような実行可能解が存在する. □

また Gabow ら [14] のアルゴリズムによって得られる式 (1.4) による定式化の実行可能解に使用されている相異なる木の数が $|V| + |E| - 2$ 以下であることから次の定理が成り立つ:

定理 4. 頂点容量制約付き有向全域木パッキング問題の目的関数値が f である実行可能解が存在するならば, 高々 $|V| + |E| - 2$ 本の相異なる木を使用した目的関数値が f である実行可能解が存在する.

Imahori ら [19] も定理 4 と同様の結果を報告している.

なお, 第 2 章と第 4 章で提案するアルゴリズムは式 (1.4) による定式化に基づいており, 式 (1.5),...,(1.12) による定式化は利用してない. 式 (1.5),...,(1.12) による定式化に対して, 有償の整数計画ソルバーである CPLEX 12.1 [1] を適用したところ, 50

[1] IBM ILOG CPLEX - Japan, http://www-06.ibm.com/software/jp/websphere/ilog/optimization/core-products-technologies/cplex/, 18, January, 2012.

頂点程度の問題例では, 比較的良い実行可能解はえられるものの, 厳密な最適解を求めることは困難であった. 第 2 章および第 4 章の計算実験で使用する 100 頂点の問題例 (2.6.1 節参照) に対しては上界 (線形緩和問題の最適解) を求めることが既に困難であり, 実行可能解を得ることができなかった.

第2章　線形緩和に基づくアルゴリズム

2.1　序論

本章では, 頂点容量制約付き有向全域木パッキング問題に対して線形緩和に基づくアルゴリズムを提案する. 線形緩和とは整数計画問題から整数制約を取り除くことであり, 線形緩和した問題を線形緩和問題と呼ぶ. 一般的に, 線形緩和問題は元の整数計画問題に比べて解きやすい問題となり, また, 元の問題に対する有効な情報 (例えば上界など) を得ることができる.

提案アルゴリズムは問題を 2 段階に分けて解く方法である. 1 段階目ではパッキングする候補の木を生成する. 線形緩和問題に対して列生成法を適用し, 停止条件を満たすまで新たな木を生成および追加することを繰り返す. 2 段階目では 1 段階目で生成した木の候補に対して, パッキングを行う回数を決定する. 線形緩和問題を解くときに得られた情報を用いて問題の実行可能解を生成し, 貪欲アルゴリズムで解を改善する.

以下では, 2.2 節で線形緩和の詳細, 2.3 で 1 段階目のアルゴリズム, 2.4 節で 2 段階目のアルゴリズムについて述べる. 提案アルゴリズムの全体の全体像が分かるように 2.5 節でアルゴリズム全体の流れを示す. 次に, 2.6 節で提案アルゴリズムの計算実験の結果を示す. センサーネットワークに関する既存研究で使用されていたセンサー位置情報から生成した問題例と, ランダムに生成した問題例の 2 つに対して実験を行ったところ, 既存アルゴリズムよりも良い解が得られることを観測した. 最後に, 2.7 節で本章の結論を述べる.

2.2　線形緩和

1.5 節で述べたように, 整数計画問題である $P(T)$ を厳密に解くことは容易ではない. そこで, 本章では $P(T)$ に線形緩和を適用する. 線形緩和とは整数計画問題から

整数制約を取り除くことであり, 線形緩和した問題を線形緩和問題と呼ぶ. 一般的に, 線形緩和問題は元の整数計画問題に比べて解きやすい問題となる. 整数計画問題の解は線形緩和問題の解となるが, 逆は成り立たない. しかし, 線形緩和問題を解くことにより, 元の整数計画問題対する有益な情報を得ることができる. この情報を利用して発見的解法を設計することがしばしば行われる [30].

$P(T)$ から整数制約 (すなわち, $x_j \in \mathbb{Z}$) を取り除いた線形緩和問題 $LP(T)$ は次のようになる:

$$
\begin{aligned}
LP(T) \qquad & \text{maximize} \quad && \sum_{j \in T} x_j, \\
& \text{subject to} \quad && \sum_{j \in T} a_{ij} x_j \leq b_i, \quad \forall i \in V, \\
& && x_j \geq 0, \qquad\quad \forall j \in T.
\end{aligned}
$$

$T = T_{\text{all}}$ のとき, 元問題 $P(T_{\text{all}})$ に対する線形緩和問題となる. また, $LP(T)$ の最適値を $\text{OPT}_{LP(T_{\text{all}})}$ と記す.

一般的に, 線形緩和問題の最適値は元の整数計画問題の最適値に対する上界となる. すなわち, $\text{OPT}_{P(T)}$ が整数値であることに注意すると, $\lfloor \text{OPT}_{LP(T)} \rfloor$ ($\lfloor x \rfloor$ は x の床関数) は $\text{OPT}_{P(T)}$ の上界となる. ただし, $T \neq T_{\text{all}}$ のとき, $\text{OPT}_{LP(T)}$ は $\text{OPT}_{P(T_{\text{all}})}$ の上界になるとは限らないことに注意する必要がある. 便宜的に, $(x_j \mid j \in T)$ を T に含まれる全ての木に対する変数 x_j のベクトルと定義する. $LP(T)$ の任意の実行可能解 $(x_j \mid j \in T)$ に対して, $(\lfloor x_j \rfloor \mid j \in T)$ は $P(T_{\text{all}})$ の実行可能解となる. また, 任意の木集合 $T \subseteq T_{\text{all}}$ に対して, そのような解 $(\lfloor x_j \rfloor \mid j \in T)$ は元問題 $P(T_{\text{all}})$ の実行可能解でもある. 本章では, これらの有益な情報を用いた線形緩和に基づくアルゴリズムを提案する.

2.3　木生成アルゴリズム

本節では，候補となる木の集合 T を生成する 1 段階目のアルゴリズム GenInTrees を提案する．まず，提案アルゴリズムは 2.3.1 節で述べる簡単なアルゴリズムによって初期木集合を生成する．その後，線形緩和問題 $LP(T)$ を解いた情報を用いて新たな木を生成し，現在の木集合 T に追加することを繰り返し行う．各反復で生成され

る木は，現在の木集合 T に追加することによって最適値 $\mathrm{OPT}_{LP(T)}$ を改善する可能性を持つものである．そのように生成された木集合は元問題 $P(T_{\mathrm{all}})$ に対する良い解を得るために有効であると考えられる．また，この手法は一般的に列生成法と呼ばれる．2.3.2 節において木を生成する方法の詳細について，2.3.3 節においてこの繰り返し処理の停止条件について述べる．最後に GenInTrees の全体像について 2.3.4 節にまとめる．

2.3.1 初期木集合

提案アルゴリズムは列生成法を用いて木を生成するが，1 本の初期木さえあれば実行できる．しかしながら，予備実験により予め数本の木をまとめて初期木集合として与えたほうが，計算時間が短くなることが観測された．また，ランダムに生成された初期木の本数を $|V|$ 本より多く増やしても，計算時間はそれほど減少しないことを観測した．以上の考察より，本研究では初期木集合として $|V|$ 本のランダムに生成された木を用いることにした．具体的には，初期木集合 T の各木は以下のように生成した．木を構成するために選択された辺集合だけを用いて r と連結であるような頂点集合を $C \subseteq V$ とする．まず初めに，$C := \{r\}$ (木として選択された辺がない) と初期化する．次に，$\{(i,j) \in E \mid i \in V \setminus C, j \in C\}$ からランダムに辺 $e = (i,j)$ を選択し，木を構成する辺として辺 e を加え，$C := C \cup \{i\}$ と更新し，この操作を $C = V$ となる (全域木が得られる) まで繰り返す．

2.3.2 列生成法

パッキングするのに有効である木集合を生成するために列生成法を用いた．列生成法は 2.3.1 節で得られた初期木集合 $T \subseteq T_{\mathrm{all}}$ から始め，停止条件を満たすまで繰り返し新たな木を生成し，木集合 T に追加する手法である．各繰り返しで生成される木は，現在の木集合 T に追加することによって最適値 $\mathrm{OPT}_{LP(T)}$ を改善する可能性を持つものである．

どのような木を現在の木集合 T に追加するかについて議論するために，まず，現在の木集合 T の線形緩和問題 $LP(T)$ (主問題) に対する双対問題 $D(T)$ を考える．双

対問題 $D(T)$ は次のように記述でき:

$$\begin{aligned} D(T) \qquad \text{minimize} \quad & \sum_{i \in V} b_i y_i, \\ \text{subject to} \quad & \sum_{i \in V} a_{ij} y_i \geq 1, \quad \forall j \in T, \\ & y_i \geq 0, \qquad\qquad \forall i \in V. \end{aligned}$$

$T = T_{\text{all}}$ であるとき，$D(T_{\text{all}})$ は線形緩和問題 $LP(T_{\text{all}})$ に対する双対問題となる．現在の木集合 T の双対問題 $D(T)$ に対する双対最適解を $(y_i^* \mid i \in V)$ と定義する．このとき, 次の条件を満たす新たな木 $\tau \in T_{\text{all}} \setminus T$ が存在するか否かを判定する問題を考える:

$$\sum_{i \in V} a_{i\tau} y_i^* < 1. \tag{2.1}$$

この問題は双対問題 $D(T)$ の制約式を双対最適解 y^* が破っているような木 τ が $T_{\text{all}} \setminus T$ の中に存在するか否かを判定する問題である．なお, 現在の木集合に含まれているどの木 $j \in T$ も, 双対最適解 y^* に対して制約式 $\sum_{i \in V} a_{ij} y_i^* \geq 1$ を満たしているので, 全ての木の中から条件式 (2.1) を満たす木 $\tau \in T_{\text{all}}$ が存在するか否かを判定する問題としても等価である．一般的に, このような問題は価格付け問題と呼ばれる．もし, 条件式 (2.1) を満たす木 τ が存在したとすると, 現在の木集合 T に追加することによって線形緩和問題 $LP(T)$ の最適値 $\mathrm{OPT}_{LP(T)}$ が改善する可能性がある．逆に存在しないときは, 最適値 $\mathrm{OPT}_{LP(T)}$ が改善できないと結論づけることができる．すなわち, 現在の木集合 T に対して, $\mathrm{OPT}_{LP(T)} = \mathrm{OPT}_{LP(T_{\text{all}})}$ と結論づけることができる.

次に, 価格付け問題を解く方法について議論する．各辺 $(v, w) \in E$ には, 始点側と終点側にそれぞれ $t((v, w))$ と $h((v, w))$ という 2 つの消費量が与えられたが, 双対問題 $D(T)$ の双対最適解 y^* を用いて, 各辺 $(v, w) \in E$ に対して次のようなコスト $\phi((v, w))$ を定義する:

$$\phi((v, w)) := y_v^* t((v, w)) + y_w^* h((v, w)). \tag{2.2}$$

また, 各木$j \in T_{\text{all}}$の辺集合を $E_j \subseteq E$ と定義する．式 (1.1) における $a_{i\tau}$ の定義より,

(2.1) の左辺は以下のように変形できる:

$$\begin{aligned}\sum_{i\in V} a_{i\tau} y_i^* &= \sum_{i\in V} y_i^* \left\{ \sum_{e\in\delta_\tau^+(i)} t(e) + \sum_{e\in\delta_\tau^-(i)} h(e) \right\} \\ &= \sum_{(v,w)\in E_\tau} \{y_v^* t((v,w)) + y_w^* h((v,w))\} \\ &= \sum_{(v,w)\in E_\tau} \phi((v,w)).\end{aligned}$$

よって, 条件式 (2.1) は次のように書ける:

$$\sum_{(v,w)\in E_\tau} \phi((v,w)) < 1. \tag{2.3}$$

すなわち, 価格付け問題は条件式 (2.3) 満たす木 $\tau \in T_{\rm all}$ が存在するかを判定する問題となる. さらに,

$$\exists\tau \in T_{\rm all}, \sum_{(v,w)\in E_\tau} \phi((v,w)) < 1 \Longleftrightarrow \min_{\tau\in T_{\rm all}} \left\{ \sum_{(v,w)\in E_\tau} \phi((v,w)) \right\} < 1 \tag{2.4}$$

より, 価格付け問題を, 条件式 (2.3) の左辺の最小値が 1 以下であるかどうかを判定する問題に変換できる. したがって, グラフ上の辺 $(v,w) \in E$ に対してコスト $\phi((v,w))$ が与えられ, 辺の合計コスト $\sum_{(v,w)\in E_\tau} \phi((v,w))$ の最小値を与えるような木 τ を見つける問題を解くことが出来れば, 価格付け問題を解くことができる. そのような問題は一般的に最小重み根指定有向全域木問題と呼ばれる古典的な問題である. (通常, 有向全域木問題は根から流出する木として定義される. しかし, 流入する木として定義しても問題の本質が変わらないないのは明らかである.)

最小重み根指定有向全域木問題とは, 入力として有向グラフ $G = (V, E)$, 根 $r \in V$ と辺のコスト $\phi : E \to \mathbb{R}$ が与えられ, 辺の合計コストが最小であるような根から流出するような有向全域木を見つける問題である. この問題は Edmonds のアルゴリズム [8] によって $O(|E||V|)$ 時間で解くことができる. Bock [4] および Chu と Liu [7] も同様の結果を報告している. Gabow ら [13] は, フィボナッチヒープを用いることにより, 現在最良の結果である $O(|E| + |V| \log |V|)$ 時間のアルゴリズムを示した.

提案アルゴリズムでは最小重み根指定有向全域木問題を解くのに Edmonds のアルゴリズムを実装した. すなわち, Edmonds のアルゴリズムによって得られた木 τ^* に

対して, $\sum_{(v,w)\in E_{\tau^*}} \phi((v,w)) < 1$ を満たすかどうかを確認すれば価格付け問題 (2.4) を解くことができる. もし, この条件を満たすならば, 得られた木 τ^* を現在の木集合 T に追加する. このような操作を 2.3.3 節で述べる停止条件が満たされるまで繰り返す.

2.3.3　停止条件

本節では列生成法の停止条件について議論する. まず始めに, 元問題 $P(T_{\rm all})$ の最適値 $\mathrm{OPT}_{P(T_{\rm all})}$ に対する上界を得る方法を示す. (列生成法の過程で得られる線形緩和問題 $LP(T)$ の最適値 $\mathrm{OPT}_{LP(T)}$ は $P(T)$ の最適値 $\mathrm{OPT}_{P(T)}$ に対する上界であって, 元問題 $P(T_{\rm all})$ の最適値 $\mathrm{OPT}_{P(T_{\rm all})}$ に対する上界ではないことに注意する必要がある.) 次の定理は列生成法の各反復で最適値 $\mathrm{OPT}_{P(T_{\rm all})}$ の上界を得ることができることを示している.

定理 5. $\hat{y}_i$ を各頂点 $i \in V$ に対する非負実数 $(\hat{y}_i \in \mathbb{R}^+)$ とし, $\rho = \min_{j\in T_{\rm all}} \left\{\sum_{i\in V} a_{ij}\hat{y}_i\right\}$ と定義する. もし, $\rho > 0$ ならば, $\sum_{i\in V} b_i(\hat{y}_i/\rho)$ は最適値 $\mathrm{OPT}_{LP(T_{\rm all})}$ の上界である.

証明. $\mathrm{OPT}_{LP(T_{\rm all})}$ と $\mathrm{OPT}_{D(T_{\rm all})}$ を, それぞれ $LP(T_{\rm all})$ と $D(T_{\rm all})$ の最適値とする. 双対定理より, $\mathrm{OPT}_{LP(T_{\rm all})} = \mathrm{OPT}_{D(T_{\rm all})}$ が成り立つ. $\rho\ (>0)$ の定義より, 全ての木 $j \in T_{\rm all}$ に対して $\sum_{i\in V} a_{ij}\hat{y}_i \geq \rho$ が成り立つ. すなわち,

$$\sum_{i\in V} a_{ij}(\hat{y}_i/\rho) \geq 1, \qquad\qquad \forall j \in T_{\rm all}$$

が成り立つ. よって, $(\hat{y}_i/\rho \mid\ i \in V)$ は双対問題 $D(T_{\rm all})$ の実行可能解であり, その目的関数値 $\omega = \sum_{i\in V} b_i(\hat{y}_i/\rho)$ は $\mathrm{OPT}_{D(T_{\rm all})} \leq \omega$ を満たす. したがって, $\mathrm{OPT}_{LP(T_{\rm all})} = \mathrm{OPT}_{D(T_{\rm all})} \leq \omega$ が成り立つ. □

列生成法の各反復で得られた双対最適解 y^* に対して, $\rho_{y^*} = \min_{j\in T_{\rm all}} \left\{\sum_{i\in V} a_{ij}y^*_i\right\}$ とする (ρ_{y^*} は条件式 (2.3) の左辺の最小値に対応する). 定理 5 より, 目的関数値 $\omega = \sum_{i\in V} b_i(y^*_i/\rho_{y^*}) = \mathrm{OPT}_{D(T)}/\rho_{y^*} = \mathrm{OPT}_{LP(T)}/\rho_{y^*}$ を求めることで, 列生成法の各反復において最適値 $\mathrm{OPT}_{P(T_{\rm all})}$ の上界を得ることができる. 最適値 $\mathrm{OPT}_{LP(T)}$ は T に新しい木を追加する度に, 単調非減少で増加していくが, 上界値 ω は単調非増加

であるとは限らない. そこで, UB^* を今までの反復の中で最良の上界 (すなわち, 最小の上界) と定義する. 定義より,

$$\mathrm{OPT}_{LP(T)} \leq \mathrm{OPT}_{LP(T_{\mathrm{all}})} \leq \mathrm{UB}^*$$

が常に成り立つ. また, UB^* の小数部を切り捨てた $\lfloor \mathrm{UB}^* \rfloor$ も最適値 $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ の上界となる. なぜなら, 元問題 $P(T_{\mathrm{all}})$ の目的関数値は整数だからである.

以上の考察より, もし,

$$\lfloor \mathrm{UB}^* \rfloor \leq \mathrm{OPT}_{LP(T)} \tag{2.5}$$

が成り立つならば, 列生成法を最後まで (価値付け問題 (2.4) を満たす木が存在しなくなるまで) 実行しなくても, 線形緩和問題 $LP(T_{\mathrm{all}})$ によって得られる最良の上界値 $\lfloor \mathrm{OPT}_{LP(T_{\mathrm{all}})} \rfloor$ が得られたと結論づけることができる. よって, 提案アルゴリズムでは列生成法の停止条件の 1 つとして条件式 (2.5) を使用する.

もし仮に, 停止条件として条件式 (2.5) だけを使用した場合を考えると, 最適値 $\mathrm{OPT}_{LP(T_{\mathrm{all}})}$ が非常に大きいとき, 数多くの木を生成してしまい, 列生成法がなかなか停止しないおそれがある. そこで, ユーザが調整できるような追加の停止条件を

$$\frac{\mathrm{UB}^* - \mathrm{OPT}_{LP(T)}}{\mathrm{OPT}_{LP(T)}} \leq \varepsilon \tag{2.6}$$

と定める. ここで, $\varepsilon \geq 0$ は非負のパラメータであり, 条件式 (2.6) は $\mathrm{OPT}_{LP(T)}$ と上界値 UB^* の間の相対誤差が ε 以下であることを表す. ε が小さい値であるほど $\mathrm{OPT}_{LP(T_{\mathrm{all}})}$ に近い上界を求めることになる. 2.6 節の計算実験では, $\varepsilon := 0.0001$ と設定した.

予備実験において, 条件式 (2.5) または (2.6) によって列生成法が停止されたとしても, 元問題 $P(T_{\mathrm{all}})$ に対して良い解が得られた. すなわち, これらのいずれかの条件が満たされるまでに生成された木集合 T は質の高い解を得るために十分であるといえる. 詳しくは 2.6 節の計算実験の結果とともに報告する.

2.3.4 提案アルゴリズム

本節では線形緩和に基づく木集合を生成するアルゴリズムの全体像を示す. 本研究では, この木集合を生成するアルゴリズムを GenInTrees と呼ぶことにする. GenInTrees アルゴリズムを擬似コード形式で以下に示す.

Algorithm GenInTrees

Input: 有向グラフ $G = (V, E)$, 根 $r \in V$, 各辺上の始点側と終点側の消費量 $t : E \to \mathbb{R}_+$ と $h : E \to \mathbb{R}_+$, 各頂点 $i \in V$ の容量 $b_i \in \mathbb{R}_+$, パラメータ ε.

Output: 木集合 T, 各反復で線形緩和問題 $LP(T)$ を解いたときに得られた最適解の集合 S, 上界値 $\lfloor \mathrm{UB}^* \rfloor$.

1: ランダムに生成した $|V|$ 本の初期木を木集合 T とする. また, $\mathrm{UB}^* := +\infty$, $S := \emptyset$, および全ての木 $j \in T$ に対して $x_j^{\max} := 0$ とする.

2: 現在の木集合 T に対する線形緩和問題 $LP(T)$ を解く. そのとき得られた最適値を $\mathrm{OPT}_{LP(T)}$, (主問題の) 最適解を $(x_j^* \mid j \in T)$, 双対最適解を $(y_i^* \mid i \in V)$ とする.

3: $S := S \cup \{(x_j^* \mid j \in T)\}$ とする.

4: グラフ G 上の全ての辺 $(v, w) \in E$ に対して, 辺コスト $\phi((v, w)) := y_v^* t((v, w)) + y_w^* h((v, w))$ を定義する.

5: 辺コスト ϕ に対して Edmonds のアルゴリズムを実行する. そのとき得られた最小合計コストを持つ木を τ, その合計コストを ρ とする.

6: もし $\mathrm{OPT}_{LP(T)}/\rho < \mathrm{UB}^*$ を満たすならば, $\mathrm{UB}^* := \mathrm{OPT}_{LP(T)}/\rho$ と更新する.

7: もし $\lfloor \mathrm{UB}^* \rfloor \leq \mathrm{OPT}_{LP(T)}$ または $(\mathrm{UB}^* - \mathrm{OPT}_{LP(T)})/\ \mathrm{OPT}_{LP(T)} \leq \varepsilon$ を満たすならば, 木集合 T, 各反復で線形緩和問題 $LP(T)$ を解いたときに得られた最適解の集合 S, 上界値 $\lfloor \mathrm{UB}^* \rfloor$ を出力して停止する. そうでなければ, $T := T \cup \{\tau\}$ と更新したのち 2 に戻る.

GenInTrees アルゴリズムでは, 実際には各反復で線形緩和問題 $LP(T)$ を解いたときに得られた最適解の集合 S を出力する. この最適解の集合 S は GenInTrees アルゴリズムを実行する上では必要なものではないが, 次の節で示す 2 段階目の木をパッキングするアルゴリズムにおいて初期解を生成するために使用される. その詳細については 2.4.1 節で述べる.

2.4 木パッキングアルゴリズム

前節では, 線形緩和に基づく 1 段階目の木を生成するアルゴリズム GenInTrees を提案した. 本節では, 2 段階目に対応する, GenInTrees アルゴリズムによって生成された木をグラフ上にパッキングする貪欲アルゴリズム PackInTrees を提案する. すなわち, 各木 $j \in T$ のパッキング回数 x_j を決定するアルゴリズムを提案する. 提案する貪欲アルゴリズムは初期実行可能解 $(x_j^{(0)} \mid j \in T)$ を必要とする. 全ての木 $j \in T$ に対して, $x_j^{(0)} := 0$ と設定された解 (全ての木に対してパッキングを 1 回も行っていないような解) を初期解とすることも可能であるが, PackInTrees アルゴリズムを適用してもあまり良い解が得られない傾向がある. そこで本研究では, 2.4.1 節で述べるように, 初期解として解集合 S に含まれる線形緩和問題の最適解の小数部を切り捨てたものを使用する. PackInTrees アルゴリズムは繰り返し各木のパッキング回数を増やす構築型のアルゴリズムである. 具体的にはそれぞれの反復において, 局所的な評価値を用いて 1 つの木のパッキング回数を増加させる. 各木 $j \in T$ の評価値には, 木 j 以外の他の木のパッキング回数を固定にした状態で, 容量制約 (1.2) を破らないという条件の下で現在のパッキング回数 x_j からさらにパッキングできる最大回数 Δ_j を用いる. 評価値の詳細については 2.4.2 節で述べる. 提案アルゴリズムの基本となるアイデアは単純である. 全ての木 $j \in T$ の中で最も大きい評価値 Δ_j を持つ木 j^* に対する現在のパッキング回数 x_{j^*} を 1 つだけ増やすという操作を全ての木 $j \in T$ が $\Delta_j = 0$ を満たすまで (パッキングできなくなるまで) 繰り返す. しかし, このアイデアをそのまま実装すると非常に大きな計算時間を必要とするので好ましくない. これを改善するための効率的な実装方法を 2.4.3 節で述べる. また, アルゴリズムの高速化手法とデータ構造について 2.4.4 節で議論する. 最後に PackInTrees の全体像について 2.4.5 節にまとめる.

2.4.1　初期解

提案する貪欲アルゴリズム PackInTrees は初期実行可能解 $(x_j^{(0)} \mid j \in T)$ を必要とする．全ての木 $j \in T$ に対して, $x_j^{(0)} := 0$ と設定された解も初期解として許されるが, そのような初期解を使用すると提案アルゴリズムは良い解を出力できないことを予備実験において観測した．一方で, 良い初期解を与えると良い実行可能解が得られることも観測した．2.2 節で述べたように, 任意の木集合 $T \subseteq T_{\text{all}}$ に対して, 線形緩和問題 $LP(T)$ における任意の実行可能解$(x_j \mid j \in T)$の小数部を切り捨てた解$(\lfloor x_j \rfloor \mid j \in T)$は元問題 $P(T_{\text{all}})$ の実行可能解となる．この事実から, GenInTrees アルゴリズムが出力した解集合 S を使用する．解集合 S に含まれる線形緩和問題の最適解の小数部を切り捨てて得られた実行可能解の集合を S' と定義する．すなわち,

$$S' = \{(\lfloor x_j^* \rfloor \mid j \in T) \mid (x_j^* \mid j \in T) \in S\} \tag{2.7}$$

である．貪欲アルゴリズムの初期実行可能解として S' の全ての要素を使用すると良い実行可能解が得られることが期待できる．しかしながら, S' の全ての要素を初期実行可能解として使用すると計算時間が大きくなってしまう．また, S' 内の全ての実行可能解が良い解であるとは限らない．そのため提案アルゴリズムでは S' 内の解より目的関数値の良いものから順に ℓ 個の実行可能解を選び, それらを初期解として使用する．もし同じ目的関数値をもつ実行可能解が複数存在したときは, GenInTrees アルゴリズムによってより後に生成された解を優先する．ここで ℓ は初期解として使用する解の数を調整するパラメータである．パラメータ ℓ をどの程度の値にするのが適切であるかについては 2.6 節で議論する．

2.4.2　木の評価基準

本節では貪欲アルゴリズムに使用する各木 $j \in T$ の評価基準について議論する．提案アルゴリズムでは, 現在の各頂点の残り容量に対して, 着目している木だけをパッキングしたときの最大回数を評価基準として使用する．$(x_j \mid j \in T)$ を元問題 $P(T_{\text{all}})$ の任意の実行可能解とすると, 各頂点 $i \in V$ の残り容量 $\bar{b}_i$ は次のように定義される:

$$\bar{b}_i = b_i - \sum_{j \in T} a_{ij} x_j. \tag{2.8}$$

これより, 各木 $j \in T$ だけをパッキングしたときの最大回数 Δ_j は次のように定義される:

$$\Delta_j = \min_{i \in V} \left\lfloor \frac{\bar{b}_i}{a_{ij}} \right\rfloor. \tag{2.9}$$

提案アルゴリズムでは Δ_j を木の評価基準として用いる. Δ_j の大きい木の方がより多くパッキングできるため, Δ_j の大きい木ほど有望な木であると期待できる.

2.4.3 効率的な実装法

貪欲アルゴリズムの基本的なアイデアは, 全ての木 $j \in T$ の中で最も大きい評価値 Δ_j を持つ木 j^* に対する現在のパッキング回数 x_{j^*} を 1 つだけ増やすという操作を, 全ての木 $j \in T$ が $\Delta_j = 0$ となるまで (パッキングできなくなるまで) 繰り返すことである. パッキング回数 x_{j^*} を 1 つずつ増やす度に残り容量 $\bar{b}$ が減少し, 各木 $j \in T$ の評価値 Δ_j も減少するが, 評価値 Δ_j の減少量は木によって異なるので, パッキング対象の木 (最も大きい評価を持つ木 j^*) は各反復ごとに変化する可能性がある.

このアルゴリズムの基本的なアイデアをそのまま実装するのは効率的でない. すなわち, 連続した反復で最も大きい評価値 Δ_j を持つ木 j^* が変化しない場合には, そのような木を 1 つずつパッキングするのは非効率である. そこで, 現在の残り容量 $\bar{b}$ における, 最も大きい評価値 Δ_j を持つ木 j^* のパッキング回数 x_{j^*} をまとめていくつか増やす. 具体的には, 現在のパッキング対象の木 j^* が全ての木 $j \in T$ の中で最も大きい評価値 Δ_j を持たなくなるまでパッキング回数 x_{j^*} を増やす.

現在のパッキング対象の木 j^* 以外の各木 $j \in T \setminus \{j^*\}$ に対して, パッキング回数 x_{j^*} を増やしたときに, パッキング対象の木 j^* の評価値 Δ_{j^*} が木 j の評価値 Δ_j 未満となるのに必要な木 j^* のパッキング回数 x_{j^*} の最小増加量を q_j と定義する. q_j はパッキング対象の木 j^* の現在のパッキング回数 x_{j^*} に対する増加量であるので,

$$0 \leq q_j \leq \Delta_{j^*} \tag{2.10}$$

を満たさなければならない. また, 全ての頂点 $i \in V$ に対して, q_j は以下の条件も満たさなければならない:

$$\left\lfloor \frac{\bar{b}_i - a_{ij^*} q_j}{a_{ij}} \right\rfloor > \Delta_{j^*} - q_j. \tag{2.11}$$

なお, 式 (2.10) を満たし, しかも全ての頂点 $i \in V$ に対して式 (2.11) を満たすような q_j が存在しない (x_{j^*} の増加量をどのように変化させても, パッキング対象の木 j^* の評価値 Δ_{j^*} が木 j の評価値 Δ_j 未満とならない) 場合も存在する. 式 (2.11) の右辺は整数値であるので, 式 (2.11) は次式と等価である:

$$\frac{\bar{b}_i - a_{ij^*} q_j}{a_{ij}} - 1 \geq \Delta_{j^*} - q_j. \tag{2.12}$$

全ての頂点 $i \in V$ に対して式 (2.12) が満たされる必要十分条件は以下の式が満たされる場合である:

$$\begin{cases} \left\lfloor \dfrac{\bar{b}_i - a_{ij}(\Delta_{j^*} + 1)}{a_{ij^*} - a_{ij}} \right\rfloor \geq q_j, & (a_{ij^*} > a_{ij}), \quad (2.13) \\ \left\lceil \dfrac{a_{ij}(\Delta_{j^*} + 1) - \bar{b}_i}{a_{ij} - a_{ij^*}} \right\rceil \leq q_j, & (a_{ij^*} < a_{ij}), \quad (2.14) \\ \left\lfloor \dfrac{\bar{b}_i}{a_{ij}} \right\rfloor \geq \Delta_{j^*}, & (a_{ij^*} = a_{ij}). \quad (2.15) \end{cases}$$

ここで, $\lceil x \rceil$ は x の天井関数を表す. q_j が式 (2.10) かつ全ての頂点 $i \in V$ に対して式 (2.13) または式 (2.14) を満たす全ての整数値の集合を $Q_j \subseteq \mathbb{Z}$ とする. なお, 式 (2.15) は q_j が取り得る整数値の範囲を制限しないので, Q_j を定義する際に考慮する必要はない. もし, $Q_j = \emptyset$, またはある頂点 $i \in V$ で式 (2.15) が満たされない場合は, q_j をどのように変化させても木 j の評価値 Δ_j がパッキング対象の木 j^* の評価値 Δ_{j^*} よりも大きくなることはない. 便宜上, このような場合は $q_j = +\infty$ と設定する. そうでない場合は, $q_j = \min(Q_j)$ だけパッキング回数 x_{j^*} を増やすことによって, 木 j の評価値 Δ_j がパッキング対象の木 j^* の評価値 Δ_{j^*} よりも大きくなる.

パッキング対象の木 j^* 以外の全ての木 $j \in T \setminus \{j^*\}$ に対する q_j の最小値を q とする. すなわち, $q = \min_{j \in T \setminus \{j^*\}} \{q_j\}$ である. もし, q が有限ならば, パッキング対象の木 j^* のパッキング回数 x_{j^*} を q だけ増やすことによって, ある木 $j \in T \setminus \{j^*\}$ の評価値 Δ_j がパッキング対象の木 j^* の評価値 Δ_{j^*} よりも大きくなる. (パッキング回数 x_{j^*} を q 未満だけ増加した場合は, 他の木 $j \in T \setminus \{j^*\}$ の評価値 Δ_j がパッキング対象の木 j^* の評価値 Δ_{j^*} よりも大きくなることはない.) そうでなければ (すなわち, $q = +\infty$ のとき), パッキング対象の木 j^* 以外の全ての木 $j \in T \setminus \{j^*\}$ の中に, パッキング対象の木 j^* のパッキング回数 x_{j^*} を増やす間に評価値 Δ_{j^*} を超えるような木が存在しないことを意味する.

貪欲アルゴリズムの効率的な実装方法では, 各反復で全ての木 $j \in T$ の中で最も大きい評価値 Δ_j を持つ木 j^* の現在のパッキング回数 x_{j^*} を 1 つだけ増やすのではなく, 代わりに最も大きい評価値 Δ_j を持つ木 j^* のパッキング回数 x_{j^*} を増加させることのできる量 q を求め, q が有限ならば x_{j^*} をまとめて q 増やし, そうでないなら x_{j^*} を最大まで増やす (すなわち, Δ_{j^*} 増やす). このような操作を全ての木 $j \in T$ が $\Delta_j = 0$ となるまで (パッキングできなくなるまで) 繰り返す.

2.4.4 高速化手法とデータ構造

前節では, パッキング回数を 1 回ずつ増やすという操作を, まとめて q 増やす方法に改善した. しかしながら, 各反復ごとに (最も大きい評価値 Δ_j を持つ木 j^* をパッキングする度に), 全ての木 $j \in T$ に対して評価値 Δ_j と増加量 q_j を計算する必要がある. 本節では, 評価値 Δ_j の上界値をソートした配列を保持することによって, それらの計算を省略する高速化手法を提案する.

$\mathfrak{D}$ を $|T|$ 個の要素を持つ降順にソートされた配列とし, $j_k \in T$ を $\mathfrak{D}$ の k 番目の要素に対応する木とする. この配列は, $\Delta_{j_1} = \mathfrak{D}[1]$, 全ての $k \in \{2, \ldots, |T|\}$ に対して $\Delta_{j_k} \leq \mathfrak{D}[k]$ という性質を満たす. すなわち, $\mathfrak{D}$ は評価値 Δ_j の上界値を降順にソートした配列である. また, $\mathfrak{D}$ の先頭の木 j_1 は全ての木 $j \in T$ の中で最も大きい評価値 Δ_j を持つ木である. すなわち, 現在の反復において $j_1 = j^*$ がパッキング対象の木となる.

貪欲アルゴリズムの 1 回目の反復においては, 全ての木 $j \in T$ に対して式 (2.9) より評価値 Δ_j を求め, それらを降順にソートして $\mathfrak{D}$ の初期値とする. すなわち, 1 回目の反復では, 全ての $k \in \{1, \ldots, |T|\}$ に対して $\Delta_{j_k} = \mathfrak{D}[k]$ である. 2 回目以降の反復では, 評価値 Δ_j の真の値を全ての木 $j \in T$ に対しては計算しない. よって, 全ての $k \in \{1, \ldots, |T|\}$ に対して $\Delta_{j_k} = \mathfrak{D}[k]$ であるとは限らない.

まず初めに, $\mathfrak{D}$ の情報を用いると, 全ての木 $j \in T$ に対して増加量 q_j を計算をしなくても q を計算できることを示す. 現在の反復において $j_1 = j^*$ がパッキング対象の木であり, $\mathfrak{D}$ のインデックス順 (すなわち, $k = 2, 3, \ldots$ の順番) に, 式 (2.10), 式 (2.13), 式 (2.14) および式 (2.15) より増加量 q_{j_k} を求めていくものとする. また, q_{j_k} の計算は k' (≥ 2) 番目まで終了したと仮定し, $q^{\min}_{j_{k'}}$ をそこまでに計算した q_{j_k} の中での最小値とする. すなわち, $q^{\min}_{j_{k'}} = \min_{k \in [2,k']} q_{j_k}$ である. $q^{\min}_{j_{k'}}$ が有限であれば, パッキ

ング回数 x_{j_1} を $q^{\min}_{j_{k'}}$ 増やしたときに, $j_2, j_3, \ldots, j_{k'}$ の中の少なくとも 1 つの木の評価値 Δ_{j_k} がパッキング対象の木の評価値 Δ_{j_1} よりも大きくなる. ここで, k' 番目まで計算が終了したときに,

$$\mathfrak{D}[k'+1] \leq \Delta_{j_1} - q^{\min}_{j_{k'}} \tag{2.16}$$

が成り立ったと仮定する. $\mathfrak{D}$ はソートされていたので, 全ての $k \geq k'+1$ に対して $\mathfrak{D}[k] \leq \Delta_{j_1} - q^{\min}_{j_{k'}}$ が成り立つ. 定義より, 全ての $k \in \{1, \ldots, |T|\}$ に対して $\Delta_{j_k} \leq \mathfrak{D}[k]$ が成立しており, また, パッキング回数 x_{j_1} を増加させても, 任意の木 $j_k \in T$ に対して評価値 Δ_{j_k} は増加しない. よって, 全ての $k \geq k'+1$ に対して, $q^{\min}_{j_{k'}}$ よりも多くパッキング回数 x_{j_1} を増加させない限り, Δ_{j_k} は Δ_{j_1} を超えることはない. すなわち, 全ての $k \geq k'+1$ に対して $q_{j_k} > q^{\min}_{j_{k'}}$ を意味する. したがって, もし k' 番目まで計算が終了したときに条件式 (2.16) が成り立つ場合は, $q^{\min}_{j_{k'}} = q = \min_{j \in T \setminus \{j^*\}} q_j$ が成立し, 全ての $k \geq k'+1$ に対して q_{j_k} の計算を省略できる.

なお, 条件式 (2.16) は $q^{\min}_{j_{k'}} = +\infty$ である状況を考慮していない. そのため, 実際には条件式 (2.16) の代わりに次ような条件式を用いる:

$$\mathfrak{D}[k'+1] \leq \Delta_{j_1} - \min\{q^{\min}_{j_{k'}}, \Delta_{j_1}\}. \tag{2.17}$$

もし, $q^{\min}_{j_{k'}}$ が有限であれば, 条件式 (2.16) と (2.17) は等価である. なぜなら, $q^{\min}_{j_k}$ の定義より, 任意の k に対して $q^{\min}_{j_k} \leq \Delta_{j_1}$ であるため, $\min\{q^{\min}_{j_{k'}}, \Delta_{j_1}\} = q^{\min}_{j_{k'}}$ となるからである. もし, $q^{\min}_{j_{k'}} = +\infty$ ならば, 条件式 (2.17) において $\mathfrak{D}[k'+1] \leq \Delta_{j_1} - \min\{q^{\min}_{j_{k'}}, \Delta_{j_1}\} = 0$ が成立することを意味する. この場合, 全ての $k \geq k'+1$ に対して $\Delta_{j_k} = \mathfrak{D}[k] = 0$ であり, Δ_{j_k} は Δ_{j_1} を超えることない. したがって, 条件式 (2.16) と同様に, もし k' 番目まで計算が終了したときに条件式 (2.17) が成り立つ場合は, $q^{\min}_{j_{k'}} = q = \min_{j \in T \setminus \{j^*\}} q_j$ が成立し, 全ての $k \geq k'+1$ に対して q_{j_k} を計算する必要はない. q の値を計算した後は, パッキング回数 x_{j_1} を増加させ, 全ての頂点 $i \in V$ に対して, 式 (2.8) より残り容量 $\bar{b}_i$ を更新する. その後, 全ての木 $j \in T$ に対する評価値 Δ_j を更新しなければならない.

次に, 上述した増加量 q の計算を省略する手法と同様の考え方を用いて, 全ての木 $j \in T$ に対する評価値 Δ_j の更新に必要な計算を省略する方法を示す. すなわち, $\mathfrak{D}$ の更新方法について議論する. $\mathfrak{D}$ のインデックス順 (すなわち, $k = 1, 2, \ldots$ の順番) に, 式 (2.9) により評価値 Δ_{j_k} を更新していくものとする. パッキング回数 x_{j_1} を増加

させる前の評価値 Δ_{j_k} に対して, 更新された新しい評価値を $\bar{\Delta}_{j_k}$ とする. また, $\bar{\Delta}_{j_k}$ の計算は k'' 番目まで終了したと仮定し, $\bar{\Delta}^{\min}_{j_{k''}}$ をそれまでに計算した $\bar{\Delta}_{j_k}$ の中での最小値とする. すなわち, $\bar{\Delta}^{\min}_{j_{k''}} = \min_{k\in[1,k'']} \bar{\Delta}_{j_k}$ である. ここで, k'' 番目まで計算が終了したときに,

$$\mathfrak{D}[k''+1] < \bar{\Delta}^{\min}_{j_{k''}} \tag{2.18}$$

が成り立つと仮定する. $\mathfrak{D}$ はソートされていたので, 全ての $k \geq k''+1$ に対して $\mathfrak{D}[k] < \bar{\Delta}^{\min}_{j_{k''}}$ が成り立つ. このとき, アルゴリズムは全ての $k \geq k''+1$ に対して新しい評価値 $\bar{\Delta}_{j_k}$ の計算を止め, $k = 1, 2, \ldots, k''$ に対する $\mathfrak{D}[k]$ の値を新しい評価値 $\bar{\Delta}_{j_k}$ に更新する. すなわち, $k = 1, 2, \ldots, k''$ に対して $\mathfrak{D}[k] := \bar{\Delta}_{j_k}$ と設定する. その後, $\mathfrak{D}[1]$ から $\mathfrak{D}[k'']$ までの要素に対して降順にソートを行う. その結果得られた配列 $\mathfrak{D}$ は全ての要素についてソートされている. すなわち, $\mathfrak{D}[1] \geq \mathfrak{D}[2] \geq \cdots \geq \mathfrak{D}[|T|]$ を満たす. また, 評価値は単調非増加 (すなわち, 任意の $k \in \{1, \ldots, |T|\}$ に対して $\bar{\Delta}_{j_k} \leq \Delta_{j_k}$) であるので, $\bar{\Delta}_{j_1} = \mathfrak{D}[1]$, および全ての $k \in \{2, \ldots, |T|\}$ に対して $\bar{\Delta}_{j_k} \leq \mathfrak{D}[k]$ が成立する. したがって, 条件式 (2.18) が成立したとき, 全ての木 $j \in T$ に対して評価値 Δ_j を更新することなく $\mathfrak{D}$ を更新することができる. $\mathfrak{D}$ を更新した後は, 再び q の値を計算する. この操作を全ての木 $j \in T$ が $\Delta_j = 0$ となるまで, すなわち, $\mathfrak{D}[1] = 0$ となるまで繰り返す.

2.4.5 提案アルゴリズム

本節では各木のパッキング回数を決定する貪欲アルゴリズム PackInTrees の全体像を示す. 2.4.3 節で示した効率的な実装方法に, 2.4.4 節で示した高速化手法とデータ構造を含めた PackInTrees アルゴリズムの擬似コードを以下に示す.

PackInTrees アルゴリズムの最悪計算量について考察する. PackInTrees アルゴリズムの 1 回の反復において, q を求めるのに最悪 $O(|V||T|)$ 時間, 評価値 Δ_j を更新するのに最悪 $O(|V||T|)$ 時間, さらに, $\mathfrak{D}$ をソートするのに最悪 $O(|T|\log|T|)$ 時間必要である. よって, 1 回の反復全体における最悪計算量は $O(|V||T|+|T|\log|T|)$ である. PackInTrees アルゴリズムのスッテプ 9 における k の値を k', スッテプ 16 における k の値を k'' とすると, 1 回の反復全体における実際の計算量は $O(|V|(k'+k'')+k''\log k'')$ となる. 多くの反復において $k', k'' \ll |T|$ であるので, 実際の計算量は最悪の計算量

Algorithm PackInTrees

Input: 問題例 $P(T)$, $P(T)$ の実行可能解 $(x_j^{(0)} \mid j \in T)$.

Output: $P(T)$ の実行可能解 $(x_j \mid j \in T)$.

1: 全ての木 $j \in T$ に対して $x_j := x_j^{(0)}$ とし, 式 (2.8) より全ての頂点 $i \in V$ 対して残り容量 $\bar{b}_i$ を計算する.

2: 式 (2.9) より全ての木 $j \in T$ に対して評価値 Δ_j を計算し, $\mathfrak{D}[j] := \Delta_j$ とする.

3: $\mathfrak{D}$ を降順にソートする. このとき, $j_k \in T$ を $\mathfrak{D}$ の k 番目の要素に対応する木とすると, 全ての $k \in \{1, \ldots, |T|\}$ に対して $\mathfrak{D}[k] = \Delta_{j_k}$ が成り立ち, かつ $\Delta_{j_1} \geq \Delta_{j_2} \geq \cdots \geq \Delta_{j_{|T|}}$ を満たす.

4: $q^{\min} := \mathfrak{D}[1]$, $k := 2$ とする.

5: 式 (2.10) を満たし, しかも全ての頂点 $i \in V$ に対して式 (2.13) または式 (2.14) を満たす整数値 q_{j_k} 全ての集合 $Q_{j_k} \subseteq \mathbb{Z}$ を求める.

6: もし $Q_{j_k} \neq \emptyset$ が成り立ち, 頂点 $i \in V$ の中に式 (2.15) を満たさないものが存在せず, しかも $q_{j_k} = \min\{Q_{j_k}\} < q^{\min}$ を満たすならば, $q^{\min} := q_{j_k}$ とする.

7: もし $k = |T|$ ならば, 9 に進む.

8: もし $\mathfrak{D}[k+1] > \mathfrak{D}[1] - q^{\min}$ ならば, $k := k + 1$ とし, 5 に戻る.

9: $x_{j_1} := x_{j_1} + q^{\min}$ と更新する.

10: 式 (2.8) より全ての頂点 $i \in V$ に対して残り容量 $\bar{b}_i$ を再計算する.

11: $\Delta^{\min} := \mathfrak{D}[1]$, $k := 1$ とする.

12: 式 (2.9) より評価値 Δ_{j_k} を再計算し, $\mathfrak{D}[k] := \Delta_{j_k}$ と更新する.

13: もし $\Delta_{j_k} < \Delta^{\min}$ ならば, $\Delta^{\min} := \Delta_{j_k}$ とする.

14: もし $k = |T|$ ならば, 16 に進む.

15: もし $\mathfrak{D}[k+1] \geq \Delta^{\min}$ ならば, $k := k + 1$ とし, 12 に戻る.

16: $\mathfrak{D}$ の 1 番目から k 番目までの要素を降順にソートする. それに応じて, 木の並び順も修正する. すなわち, 全ての $\hat{k} \in \{1, 2, \ldots, k\}$ に対して $j_{\hat{k}}$ を修正する.

17: もし, $\mathfrak{D}[1] = 0$ ならば, 実行可能解$(x_j \mid j \in T)$を出力して停止する. そうでなければ 4 に戻る.

よりも十分小さくなる. 各反復において少なくとも1本の木がパッキングされるので, 上界値 $\mathrm{OPT}_{LP(T_{\mathrm{all}})}$ よりも多くの反復を繰り返すことはない. よって, PackInTrees アルゴリズムは任意の初期解 (全ての木 $j \in T$ に対して $x_j = 0$ であるような初期解) に対して最悪$O(\mathrm{OPT}_{LP(T_{\mathrm{all}})}(|V||T| + |T|\log|T|))$時間で動作する.

2.4.1 節で述べたように, PackInTrees アルゴリズムは式 (2.7) で定義した S' の要素を初期解に使用する. すなわち, GenInTrees アルゴリズムのある反復において生成された木集合を T' とすると, 線形緩和問題 $LP(T')$ の最適解 $(x_j^* \mid j \in T')$ の小数部を切り捨てた解 $(\lfloor x_j^* \rfloor \mid j \in T')$ が初期解として使用される. つまり, そのような解を初期解として使用した場合, $\mathrm{OPT}_{LP(T_{\mathrm{all}})} - \mathrm{OPT}_{LP(T')} + |V|$ よりも多くの反復を繰り返すことはない. したがって, $\mathrm{ITR} = \mathrm{OPT}_{LP(T_{\mathrm{all}})} - \mathrm{OPT}_{LP(T')} + |V|$ と定義すると, S' の要素を初期解に使用する場合における PackInTrees アルゴリズムの最悪計算量は$O(\mathrm{ITR}(|V||T| + |T|\log|T|))$になる.

2.5 アルゴリズムの全体像

これまでの節で, 1段階目のアルゴリズムとして GenInTrees アルゴリズム, 2段階目のアルゴリズムとして PackInTrees アルゴリズムを提案した. 本節では線形緩和に基いた提案アルゴリズムの全体像について簡単にまとめる. 提案アルゴリズムは最初に GenInTrees アルゴリズムを呼び出し, 木の候補 T と線形緩和問題の最適解の集合 S を得る. 次に, S 内の解の小数部を切り捨てて得られた実行可能解の集合 S' を生成し, S' の要素の中で目的関数値の良いものから順に ℓ 個を選択する. その後, 選んだ ℓ 個の実行可能解に対して PackInTrees アルゴリズムを適用して解の改善を行う. 最後に, PackInTrees アルゴリズムが出力した ℓ 個の実行可能解の中で, 最良の実行可能解を提案アルゴリズムの解として出力する.

2.6 計算実験

2.6.1 計算環境

計算実験には2つのタイプの問題を用いた. 1つ目はセンサーネットワークに関する研究 [17, 26] で使用されたセンサーの位置情報に基づくものである. それらの位置

情報から対称的な始点側と終点側の消費量, 頂点容量を持つ完全有向グラフを生成した. 消費量はパケットの送信時と受信時にかかる電力量, 頂点容量はセンサーの持つバッテリーの電力容量として設定した. 具体的には, Heinzelman ら [17] と Sasaki ら [26] と同様に, $E_{\mathrm{elec}} = 50$ nJ/bit, $\varepsilon_{\mathrm{fs}} = 10$ pJ/bit/m^2, $\varepsilon_{\mathrm{mp}} = 0.0013$ pJ/bit/m^4, $E_{\mathrm{DA}} = 5$ nJ/bit/signal, $l = 4200$ bit, $d_0 = 87$ m というパラメータを使用した. 頂点 (センサー) v から w の間のユークリッド距離を $d((v,w))$ と定義する. 各辺 $(v,w) \in E$ の消費量を次のように設定した:

$$t((v,w)) = \begin{cases} lE_{\mathrm{elec}} + l\varepsilon_{\mathrm{fs}}\{d((v,w))\}^2, & \text{if } d((v,w)) < d_0, \\ lE_{\mathrm{elec}} + l\varepsilon_{\mathrm{mp}}\{d((v,w))\}^4, & \text{if } d((v,w)) \geq d_0, \end{cases}$$

$$h((v,w)) = lE_{\mathrm{elec}} + lE_{\mathrm{DA}}.$$

消費量はユークリッド距離の関数であるので, 全ての辺 $(v,w) \in E$ に対して $t((v,w)) = t((w,v))$ および $h((v,w)) = h((w,v))$ が成り立つ. 根 (基地局) の頂点容量を無限 $b_r = +\infty$ J, 残りの全ての頂点 $i \in V \setminus \{r\}$ の頂点容量を $b_i = 0.5$ J と設定した. また, 基地局および全てのセンサー間で通信できるものとし, 全ての頂点対に辺を与えた (すなわち, $E = \{(v,w) \mid v \neq w, \forall v \in V, \forall w \in V\}$). 本研究では, これらのセンサーの位置情報から生成された問題例を hcb100, sfis100-1, sfis100-2, sfis100-3 と呼ぶことにする. 問題例 hcb100 は Heinzelman ら [17] で用いられたセンサーの位置情報から生成された問題例であり, sfis100-1, 2, 3 は Sasaki ら [26] で用いられたセンサーの位置情報 data1, 2, 3 からそれぞれ生成された問題である.

2 つ目はランダムに生成されたグラフである. 本研究では, これらの問題例を “rnd$n$-δ-b-(t, h or none)” と呼ぶことにする. ここで, n は根を除いた頂点数 $|V^-|$ ($V^- = V \setminus \{r\}$), δ は辺密度, b は根を除いた全ての頂点 $i \in V^-$ の容量 (根の容量は無限 $b_r = +\infty$), “t, h or none” は始点側と終点側の消費量のどちらが大きいかを表す (“t” は始点側の消費量が終点側の消費量より大きいことを意味し, “h” は終点側の消費量が始点側の消費量より大きいことを意味する. また, どちらでもない “none” である場合は始点側と終点側の消費量は同じ範囲からランダムに選ばれることを意味する). 具体的には, $n = 100$ と $\delta = 5\%, 50\%$ と $b = 10000, 100000$ である問題例を 12 個生成した. $\delta = 5\%$ (50%) である問題例は, 各頂点の出る辺の次数が頂点数 $|V^-|$ に対して 4% (40%) から 6% (60%) となるようにランダムに辺を追加した. “t” で

ある問題例は始点側と終点側の消費量をそれぞれ閉区間 $[30, 50]$ と $[3, 5]$ からランダムに選んだ整数値とした. 同様に “h” である問題例は始点側と終点側の消費量をそれぞれ閉区間 $[3, 5]$ と $[30, 50]$ からランダムに選んだ整数値とし, どちらでもない問題例はそれぞれ閉区間 $[30, 50]$ と $[30, 50]$ からランダムに選んだ整数値とした. ただし, 根に入る辺ばかりが優先的に使用されないようにするために, 根に入る全ての辺 $(\forall v \in V^-, e = (v, r))$ に対しては, 始点側の消費量を閉区間 $[300, 500]$ からランダムに選んだ整数値とした. これらの問題例は Web ページ[1]で公開されている.

提案アルゴリズムを C++言語で実装し, Dell PowerEdge T300 (CPU: Xeon X3363 2.83GHz, キャッシュ: 6MB, メモリ: 24GB) で計算実験を行った. 線形計画ソルバーはフリーで公開されている GLPK4.43[2]を使用し, 解法にはシンプレックス法を用いた.

2.6.2 計算結果

図 2.1 と 2.2 は GenInTrees アルゴリズムに問題例 sfis100-1 と rnd100-50-100000-h を入力したときの挙動を表している. 図の横軸は生成された木の数, 縦軸は目的関数値である. すなわち, 左から右へアルゴリズムの反復が進む. 各反復で木集合 T に新しい木が追加されたときの, 線形緩和問題の最適値 $\mathrm{OPT}_{LP(T)}$ と元問題 $P(T_{\mathrm{all}})$ の上界 UB^* の改善の様子を示している. 反復が進むごとに, 最適値 $\mathrm{OPT}_{LP(T)}$ と上界 UB^* の間の差は小さくなり, 最終的に非常に接近した値となる. しかし, 改善の比率は小さくなっていく. 他の問題例に対してもこのような傾向が観測される. 一般的に, このような傾向は列生成法を用いたアルゴリズムでしばしば観測される.

2.4.1 節で述べたように, S' 内の解を PackInTrees アルゴリズムの初期実行可能解として使用する. 問題例 sfis100-1 と rnd100-50-100000-h に対する S' 内の解の目的関数値と, それらの S' 内の解を入力としたときの PackInTrees アルゴリズムが出力した解の目的関数値の関係を図 2.3 と 2.4 に示す. 図の横軸が S' 内の解の目的関数値, 縦軸が PackInTrees アルゴリズムが出力した解の目的関数値を表し, 各点が入出力のペアを意味する. 図 2.3 と 2.4 より, 2 つの解の目的関数値の間に強い相関関係があることが観測できる. この傾向は他の問題例に対しても観測される. この結果か

[1] http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/ncipp/

[2] GLPK-GNU Project-Free Software Foundation (FSF), http://www.gnu.org/software/glpk/, 3, July, 2010.

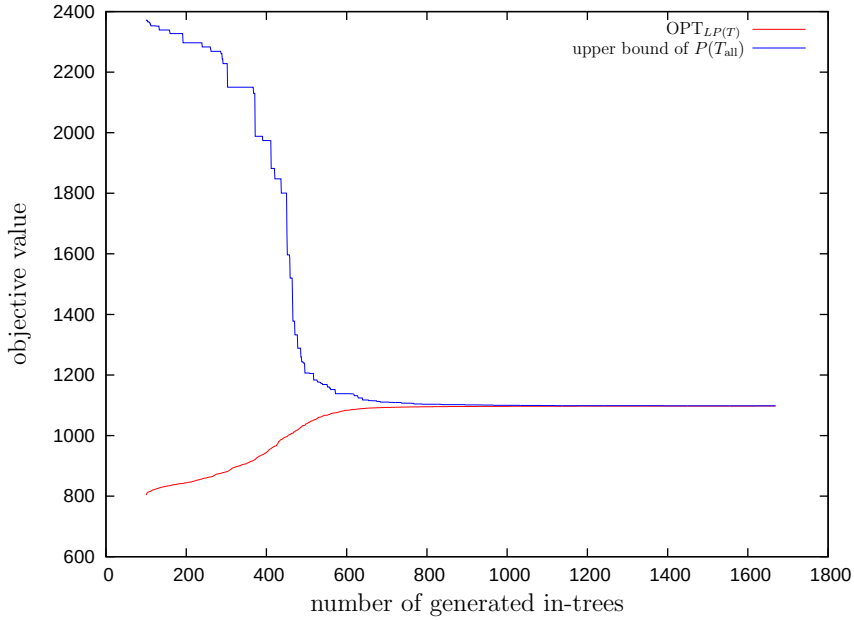


図 2.1: GenInTrees アルゴリズムに問題例 sfis100-1 を入力したときの挙動

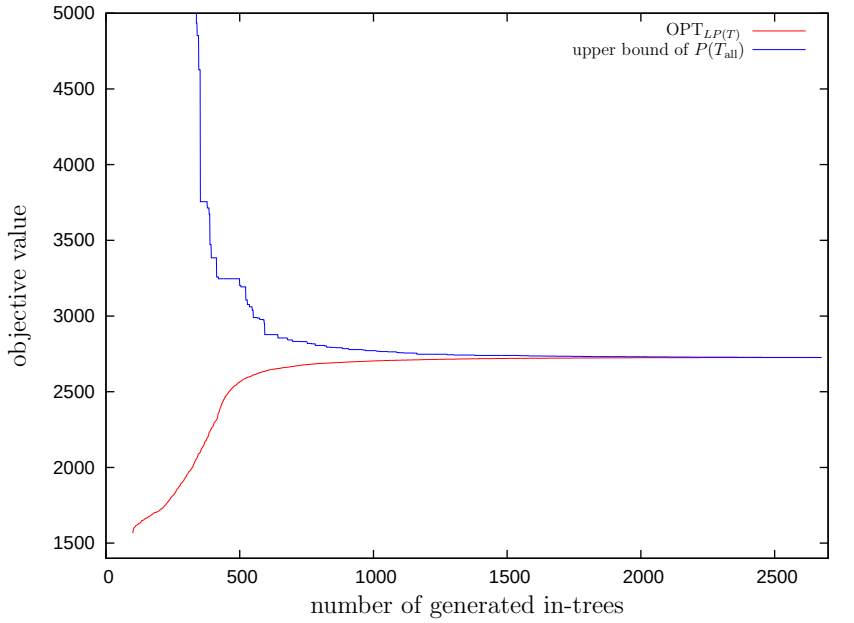


図 2.2: GenInTrees アルゴリズムに問題例 rnd100-50-100000-h を入力したときの挙動

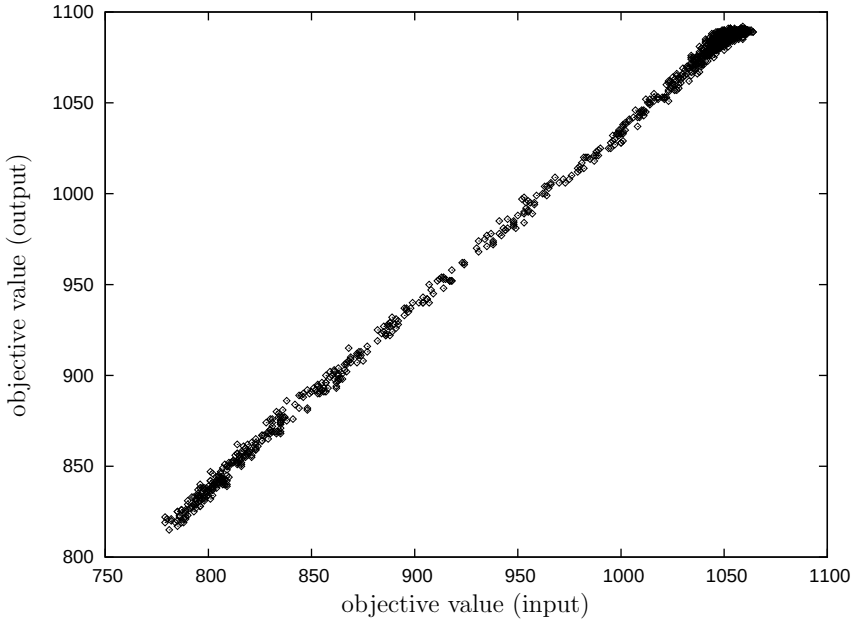


図 2.3: 問題例 sfis100-1 に対する, PackInTrees アルゴリズムの初期解 (S' 内の解) と出力解の目的関数値の関係

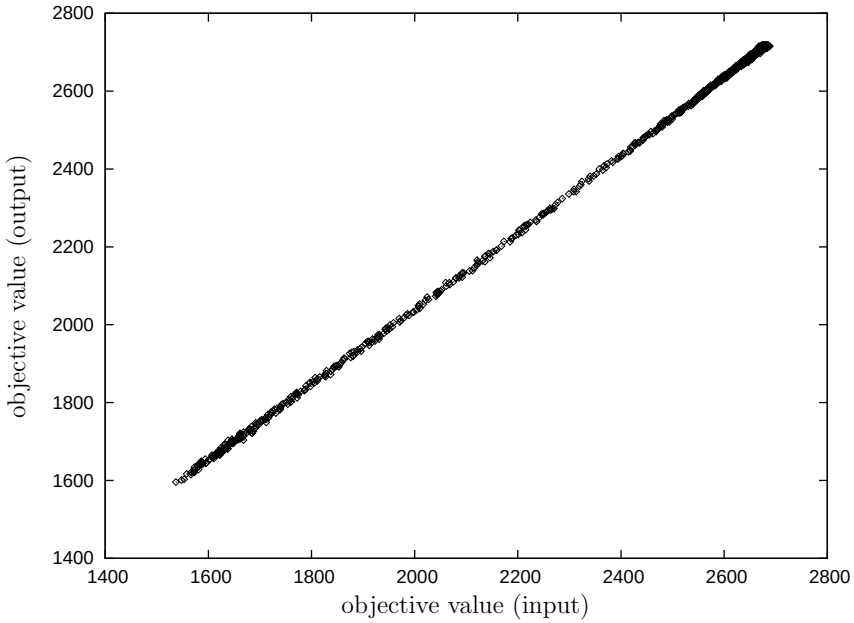


図 2.4: 問題例 rnd100-50-100000-h に対する, PackInTrees アルゴリズムの初期解 (S' 内の解) と出力解の目的関数値の関係

ら, 元問題 $P(T_{\mathrm{all}})$ に対する良い実行可能解を得るという目的のためには, 小さい目的関数値を持つ S' 内の解を PackInTrees アルゴリズムの初期解として使用することは必要ないことが分かる. よって, S' 内の解のうち目的関数値が良いものから上位数%の解を PackInTrees アルゴリズムの初期解としてして使用だけで十分である. そのため, 提案アルゴリズムでは, S' 内の解のうち上位 ℓ 個のみを PackInTrees アルゴリズムの初期解として使用する.

次に, パラメータ ℓ による提案アルゴリズムの影響について考察する. 2.6.1 節で述べた問題例に対して, $\ell = 1, \lfloor 0.01|T| \rfloor, \lfloor 0.05|T| \rfloor, |T|$ と設定したときの提案アルゴリズムの結果を表 2.1 に示す. 最初の 3 列は, 問題例の名前, 根を除いた頂点数 $|V^-|$, 辺数 $|E|$ を表す. 列 "$|T|$" は GenInTrees アルゴリズムによって生成された木の数, 列 "UB" は GenInTrees アルゴリズムによって得られた元問題 $P(T_{\mathrm{all}})$ の最適値 $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ に対する最良の上界, 列 "$S'_{\max}$" は S' 内の解の目的関数値の最大値をそれぞれ表す. 続く残りの列は, 4 種類の ℓ の値に対して, 列 "Obj." は提案アルゴリズムが出力した目的関数値, 列 "Time (s)" は計算時間 (秒) を表す. 表 2.1 の $\ell = 1$ と $\ell = \lfloor 0.01|T| \rfloor$ の結果を比較すると, いくつかの問題例において目的関数値が改善している. $\ell = \lfloor 0.01|T| \rfloor$ と $\ell = \lfloor 0.05|T| \rfloor$ の結果の間では 2 つの問題例 sfis100-1 と rnd100-5-10000-t の目的関数値が改善している. さらに, $\ell = \lfloor 0.05|T| \rfloor$ と $\ell = \lfloor |T| \rfloor$ の結果を比較すると, 目的関数値が改善している問題例が存在しないことが観測できる. すなわち, 表 2.1 では ℓ の値を (PackInTrees アルゴリズムに入力する解の数を) $\lfloor 0.05|T| \rfloor$ 以上に増やしても目的関数値が改善しない. この結果は図 2.3 と 2.4 から得られた考察に一致する. また, $\ell = |T|$ の計算時間は $\ell = 1$ に比べて約 3 から 7 倍に, $\ell = \lfloor 0.05|T| \rfloor$ の計算時間は $\ell = 1$ に比べて約 10%から 30%増加している. 以上の考察から, 提案アルゴリズムでは $\ell := \lfloor 0.05|T| \rfloor$ と設定している. (提案アルゴリズムは ℓ の値に対して挙動が敏感に変ることはないので, ℓ の値を精密にチューニングすることは必要でない.)

最後に, 提案アルゴリズムによって得られた解と Sasaki ら [26] のアルゴリズムによって得られた解を比較検討する. ただし, 彼らのアルゴリズムは基地局が全てのセンサーからパケットを受信できなくなってもパケットを収集するように設計されている. バッテリーがなくなってしまったセンサーが少なとも 1 つ存在する場合にそのような状況が発生する. そのため, 彼らはバッテリーが残っているセンサーの数に対

表 2.1: パラメータ ℓ が提案アルゴリズムの結果に及ぼす影響

Instance name	$\lvert V^- \rvert$	$\lvert E \rvert$	GenInTrees		$S'_{\max}$	$\ell = 1$		$\ell = \lfloor 0.01\lvert T \rvert \rfloor$		$\ell = \lfloor 0.05\lvert T \rvert \rfloor$		$\ell = \lvert T \rvert$	
			$\lvert T \rvert$	UB	Obj.	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)
hcb100	100	10100	2752	1124	1095	1121	86	1121	89	1121	97	1121	302
sfis100-1	100	10100	1669	1097	1064	1089	32	1090	33	1092	37	1092	115
sfis100-2	100	10100	1700	1097	1065	1090	34	1090	35	1090	39	1090	127
sfis100-3	100	10100	1378	1101	1067	1095	23	1095	24	1095	26	1095	68
rnd100-5-10000-h	100	473	2093	225	195	215	43	219	44	219	51	219	182
rnd100-5-10000-t	100	473	1227	217	183	205	15	206	15	207	18	207	69
rnd100-5-10000	100	473	2041	128	100	123	43	123	43	123	50	123	188
rnd100-5-100000-h	100	473	2515	2251	2211	2243	59	2245	62	2245	76	2245	343
rnd100-5-100000-t	100	473	1227	2173	2133	2163	15	2163	15	2163	18	2163	67
rnd100-5-100000	100	473	2496	1283	1248	1278	60	1278	63	1278	77	1278	352
rnd100-50-10000-h	100	4938	2078	272	244	265	46	267	48	267	55	267	227
rnd100-50-10000-t	100	4938	1688	270	234	256	23	256	25	256	29	256	110
rnd100-50-10000	100	4938	2373	149	124	144	59	145	62	145	74	145	339
rnd100-50-100000-h	100	4938	2675	2726	2688	2716	73	2718	77	2718	98	2718	518
rnd100-50-100000-t	100	4938	1860	2701	2661	2686	27	2688	28	2688	36	2688	145
rnd100-50-100000	100	4938	2804	1498	1466	492	82	1493	88	1493	109	1493	623

して, パケットを収集できた最大回数を報告している. 本研究では全域木をパッキングした回数を目的関数値としているので, 彼らの報告した結果の中から, 全センサーのバッテリーが残っている状態でパケットを収集できた最大回数を比較に使用する.

表 2.2 は, 提案アルゴリズムによって得られた解と Sasaki ら [26] のアルゴリズムによって得られた解の比較結果である. 最初の 5 列は表 2.1 の最初の 5 列と同じものである. 次の 4 列は $\ell = \lfloor 0.05|T| \rfloor$ であるときの提案アルゴリズムの結果を示しており, 左から順に, 目的関数値, UB と Obj. の間のギャップ (すなわち, $((\mathrm{UB} - \mathrm{Obj.})/\mathrm{UB}) \times 100\%$), 最良解で使用された木の数 $|T_{\mathrm{pos}}(x_{\mathrm{best}})|$ (ここで, $T_{\mathrm{pos}}(x) = \{j \in T \mid x_j > 0\}$ であり, x_{best} は提案アルゴリズムで得られた最良解を意味する), 計算時間 (秒) を表す. 列 “Sasaki et al. (2007)” は, 彼らが報告した全センサーのバッテリーが残っている状態でパケットを収集できた最大回数を表す. ただし, “–” は結果が得られていないことを意味する. これは, 彼らの実装コードが平面上のユークリッド距離に基づいて消費量が定まる問題例に特化されており, ランダムに生成したグラフに対応していないためである. そこで, 計算結果を補うために彼らのアルゴリズムを新たに実装した. 最後の列 SFIS は Sasaki ら [26] のアルゴリズムを新たに実装したものに対して, 制限時間を 600 秒制としたときの結果を表す.

新たに実装したものによる結果と彼らの実装による結果が異なっているのは, 細かな部分で実装が多少異なっていることが原因であると考えられる. しかしながら, sfis100-1 と 2 に対して新たな実装 SFIS の方が元の実装に比べて良い解を与えており, 僅かに SFIS の方が優れていると考えられる.

表 2.2 より, 提案アルゴリズムは Sasaki ら [26] のアルゴリズムの元の実装と新たな実装 SFIS に対して, rnd100-5-10000-t を除く全ての問題例において, より良い解をえていることが観測できる. また, 目的関数値が 1000 を超えるような問題例に対しては, 上界と目的関数値の間のギャップは非常に小さい. 一方, 目的関数値が小さい問題例に対してはギャップは大きくなる. 興味深い結果として, 全ての問題例に対して, 上界と目的関数値の絶対的な差は小さくそれほど変わらない (UB − Obj. は 3 から 14 までの範囲に収まっている). また, 提案アルゴリズムの出力した最良解 x_{best} に使用される木の数が, 頂点数 $|V^-|$ を超えることは少ないことも観測できる.

表 2.2: 既存アルゴリズムとの比較結果

Instance name	$\|V^-\|$	$\|E\|$	GenInTrees		Proposed Algorithm ($\ell = \lfloor 0.05\|T\| \rfloor$)				Sasaki et al. (2007)	SFIS
			$\|T\|$	UB	Obj.	Gap (%)	$\|T_{\mathrm{pos}}(x_{\mathrm{best}})\|$	Time (s)		
hcb100	100	10100	2752	1124	1121	0.27	69	97	–	931
sfis100-1	100	10100	1669	1097	1092	0.46	77	37	961	1033
sfis100-2	100	10100	1700	1097	1090	0.64	74	39	969	1032
sfis100-3	100	10100	1378	1101	1095	0.54	75	26	1022	1021
rnd100-5-10000-h	100	473	2093	225	219	2.67	69	51	–	160
rnd100-5-10000-t	100	473	1227	217	207	4.61	74	18	–	210
rnd100-5-10000	100	473	2041	128	123	3.91	53	50	–	99
rnd100-5-100000-h	100	473	2515	2251	2245	0.27	104	76	–	1611
rnd100-5-100000-t	100	473	1227	2173	2163	0.46	93	18	–	2108
rnd100-5-100000	100	473	2496	1283	1278	0.39	85	77	–	1003
rnd100-50-10000-h	100	4938	2078	272	267	1.84	71	55	–	211
rnd100-50-10000-t	100	4938	1688	270	256	5.19	82	29	–	238
rnd100-50-10000	100	4938	2373	149	145	2.68	51	74	–	120
rnd100-50-100000-h	100	4938	2675	2726	2718	0.29	102	98	–	2293
rnd100-50-100000-t	100	4938	1860	2701	2688	0.48	106	36	–	2414
rnd100-50-100000	100	4938	2804	1498	1493	0.33	84	109	–	1295

2.7 結論

本章では, 頂点容量制約付き有向全域木パッキング問題に対する線形緩和に基づく発見的解法を提案した. 提案アルゴリズムは2段階で構成されており, まず候補となる木の集合Tを生成し, 次に生成した各木$t \in T$に対してパッキング回数を決定する. 1段階目では, 線形緩和問題$LP(T)$に列生成法を適用し, 繰り返し新たな木を生成してTに追加する方法を提案した. その中で, これを実現するために解く必要のある価格付け問題が, 最小重み根指定有向全域木問題という古典的な問題と等価であることを示した. 2段階目は, 第1段階で生成した線形緩和問題$LP(T)$の最適解xを修正して実行可能解を作り, 貪欲アルゴリズムによって解を改善する. この貪欲アルゴリズムに対して効率的な実装方法を提案し, データ構造を工夫することにより高速化した. 提案アルゴリズムは既存アルゴリズムよりも良い解を出力し, 上界と目的関数値のギャップが非常に小さいことを観測した. 本章で提案したアルゴリズムは頂点容量制約付き有向全域木パッキング問題に対して有効であると結論づけられる.

第3章　線形緩和に基づくアルゴリズムの近似精度

3.1　序論

本章では, 頂点容量制約付き有向全域木パッキング問題に対する線形緩和に基づくアルゴリズムの近似精度について議論する. 第 2 章で提案したアルゴリズムだけではなく, 線形緩和問題 $LP(T_{\mathrm{all}})$ の最適解を使用するアルゴリズムに対する一般的な議論である.

まず, 近似精度の議論のために 3.2 節で線形緩和に基づくアルゴリズムの一般化を行い, 定義されたアルゴリズムの性質について議論する. 3.3 節では, 最適値 $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ が頂点数 $|V|$ の α 倍 $(\alpha > 1)$ である問題例に対して, 線形緩和に基づくアルゴリズムが $(1-1/\alpha)$ 近似アルゴリズムであることを示す. 次の 3.4 節では, 線形緩和に基づくアルゴリズムの近似精度の限界について議論を行う. 最後に, 3.5 節で結論を述べる.

3.2　準備

本節では, 線形緩和に基づくアルゴリズムの一般化を行い, 定義されたアルゴリズムの性質について議論する. まず, 線形緩和に基づくアルゴリズムによって得られる木集合が持つべき性質を定義する. T_{lp} を次のような性質を持った木集合と定義する:

1. $\mathrm{OPT}_{LP(T_{\mathrm{lp}})} = \mathrm{OPT}_{LP(T_{\mathrm{all}})}$.
2. 任意の木 $j \in T_{\mathrm{lp}}$ に対して, $\mathrm{OPT}_{LP(T_{\mathrm{lp}}\setminus\{j\})} < \mathrm{OPT}_{LP(T_{\mathrm{all}})}$ (すなわち, $\mathrm{OPT}_{LP(T_{\mathrm{all}})}$ を達成するのに冗長である木が T_{lp} に含まれていない).

全ての木の集合 T_{all} の要素数は指数的なサイズになるが, $|T_{\mathrm{lp}}| \leq |V|$ を満たすような木集合 T_{lp} と, それに対応する線形緩和問題 $LP(T_{\mathrm{lp}})$ の最適解 $(\bar{x}_j \mid j \in T_{\mathrm{lp}})$ を入力サイズに対して多項式時間で見つけられることが知られている [27].

2.2 節で述べたように, 任意の木集合 $T \subseteq T_{\mathrm{all}}$ の線形緩和問題 $LP(T)$ における任意の実行可能解 $(x_j \mid j \in T)$ に対して, $(\lfloor x_j \rfloor \mid j \in T)$ は元問題 $P(T_{\mathrm{all}})$ の実行可能解である. この性質を用いて, 簡単な線形緩和に基づくアルゴリズム SimpleLPBaseAL を以下の疑義コードのように定義する. この SimpleLPBaseAL アルゴリズムが出

Algorithm SimpleLPBaseAL

Input: 有向グラフ $G = (V, E)$, 根 $r \in V$, 各辺上の始点側と終点側の消費量 $t : E \to \mathbb{R}_+$ と $h : E \to \mathbb{R}_+$, 各頂点 $i \in V$ の容量 $b_i \in \mathbb{R}_+$.

Output: 木集合 T_{lp}, 実行可能解 $(\lfloor \bar{x}_j \rfloor \mid j \in T_{\mathrm{lp}})$.

1: 木集合 T_{lp} を求める (ただし, 上述の性質 1 と 2, および $|T_{\mathrm{lp}}| \leq |V|$ を満たす).
2: 線形緩和問題 $LP(T_{\mathrm{lp}})$ の最適解 $(\bar{x}_j \mid j \in T_{\mathrm{lp}})$ を求める.
3: $(\lfloor \bar{x}_j \rfloor \mid j \in T_{\mathrm{lp}})$ を出力する.

力した目的関数値 $\sum_{j \in T_{\mathrm{lp}}} \lfloor \bar{x}_j \rfloor$ は以下の性質を満たす:

$$\begin{aligned} \mathrm{OPT}_{P(T_{\mathrm{all}})} \leq \mathrm{OPT}_{LP(T_{\mathrm{all}})} = \mathrm{OPT}_{LP(T_{\mathrm{lp}})} &= \sum_{j \in T_{\mathrm{lp}}} \bar{x}_j \\ &< \sum_{j \in T_{\mathrm{lp}}} (\lfloor \bar{x}_j \rfloor + 1) \leq |V| + \sum_{j \in T_{\mathrm{lp}}} \lfloor \bar{x}_j \rfloor. \end{aligned}$$

したがって, 次の補題が成り立つ.

補題 1. 頂点容量制約付き有向全域木パッキング問題に対する SimpleLPBaseAL アルゴリズムは $\mathrm{OPT}_{P(T_{\mathrm{all}})} - |V|$ よりも大きい目的関数値を持つ実行可能解を出力する.

次節で, この補題を用いて SimpleLPBaseAL アルゴリズムの近似精度について議論する.

3.3　近似精度

本節では, 前節で定義した SimpleLPBaseAL アルゴリズムの近似精度について議論する. 前節の補題 1 で, SimpleLPBaseAL アルゴリズムは $\mathrm{OPT}_{P(T_{\mathrm{all}})} - |V|$ よりも大きい目的関数値を持つ実行可能解を出力することを示した. もし, SimpleLPBaseAL アルゴリズムに入力された問題例の最適値 $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ が頂点数 $|V|$ の定数

倍以上である, すなわち, ある定数 $\alpha > 1$ に対して $\mathrm{OPT}_{P(T_{\mathrm{all}})} \geq \alpha|V|$ を満たすならば,

$$\sum_{j \in T_{\mathrm{lp}}} \lfloor \bar{x}_j \rfloor > \mathrm{OPT}_{P(T_{\mathrm{all}})} - |V| \geq \mathrm{OPT}_{P(T_{\mathrm{all}})} - \frac{\mathrm{OPT}_{P(T_{\mathrm{all}})}}{\alpha} = \left(1 - \frac{1}{\alpha}\right) \mathrm{OPT}_{P(T_{\mathrm{all}})}$$

が成り立つ. よって, 以下の定理が成り立つ.

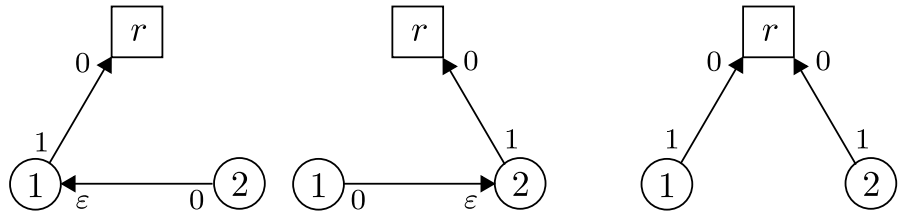
定理 6. 問題例がある定数 $\alpha > 1$ に対して $\mathrm{OPT}_{P(T_{\mathrm{all}})} \geq \alpha|V|$ を満たすならば, SimpleLPBaseAL アルゴリズムは $(1 - 1/\alpha)$ 近似アルゴリズムである. また, $\mathrm{OPT}_{P(T_{\mathrm{all}})}/|V| \to +\infty$ のとき, 近似率は 1 に収束する.

なお, 3.2節で T_{lp} の持つべき性質を定義したが, 性質1を $\mathrm{OPT}_{LP(T_{\mathrm{lp}})} \geq \lfloor \mathrm{OPT}_{LP(T_{\mathrm{all}})} \rfloor$ に変更しても, SimpleLPBaseAL アルゴリズムの近似精度に関する上述の証明と同様の議論ができる. また, 性質 2 は SimpleLPBaseAL アルゴリズムの近似精度に関する上述の議論には必要ない. (性質 2 は, アルゴリズムが $LP(T_{\mathrm{all}})$ のある最適解に必要な木以外を利用できないことを明示するものであるが, これは, 3.4 節において, この性質のために近似精度に限界があることを示すためのものである.)

第 2 章で提案したアルゴリズムおいて, 条件式 (2.5) で列生成法が停止したときの木集合を T とする. このとき, 線形緩和問題 $LP(T)$ の端点最適解 $(\bar{x}_j \mid j \in T)$ に対して $T_{\mathrm{lp}} = T_{\mathrm{pos}}(\bar{x})$ $(T_{\mathrm{pos}}(x) = \{j \in T \mid x_j > 0\})$ とすると, $\mathrm{OPT}_{LP(T_{\mathrm{lp}})} \geq \lfloor \mathrm{OPT}_{LP(T_{\mathrm{all}})} \rfloor$ が成り立つ. また, S' の中にこの解 $(\bar{x}_j \mid j \in T)$ が含まれてるので, 最終的に最良解として出力される解の目的関数値は $\sum_{j \in T} \lfloor \bar{x}_j \rfloor$ 以上である. よって, 第 2 章で提案したアルゴリズムは条件式 (2.5) で列生成法が停止したとき, $(1 - 1/\alpha)$ 近似アルゴリズムである.

3.4 近似精度の限界

前節で, SimpleLPBaseAL アルゴリズムの近似精度について述べた. 本節では, SimpleLPBaseAL アルゴリズムに対する近似精度の限界について議論する. 近似精度の限界を示すために, SimpleLPBaseAL アルゴリズムの近似精度がいくらでも悪くなってしまう問題例を与える.



(a)　　(b)

図 3.1: SimpleLPBaseAL アルゴリズムの近似精度が悪くなっていまう問題例 ($|V| = 3$, $\mathrm{OPT}_{P(T_{\mathrm{all}})} = 1$, $\mathrm{OPT}_{P(T_{\mathrm{lp}})} = 0$)

まず, 次のような (3 つの頂点を持つ) 問題例を与える:

$$
\begin{aligned}
&V = \{r, 1, 2\},\ E = \{(1, r), (2, r), (1, 2), (2, 1)\},\\
&b(1) = b(2) = 1,\ b(r) = +\infty,\\
&t((1, r)) = t((2, r)) = 1,\ h((1, r)) = h((2, r)) = 0,\\
&t((1, 2)) = t((2, 1)) = 0,\ h((1, 2)) = h((2, 1)) = \varepsilon.
\end{aligned}
$$

ここで, ε は $0 < \varepsilon \ll 1$ を満たす任意の値であり, $t((u, v))$ と $h((u, v))$ は辺 (u, v) の始点側と終点側の消費量を表す. この問題例に対して T_{lp} は一意的に定まり, $|T_{\mathrm{lp}}| = 2$ および $\mathrm{OPT}_{LP(T_{\mathrm{lp}})} = 2/(1 + \varepsilon)$ を満たす (図 3.1(a) 参照). T_{lp} 内の木は元問題 $P(T_{\mathrm{all}})$ の解に貢献しない. すなわち, 任意の木 $j \in T_{\mathrm{lp}}$ はグラフ上に 1 つもパッキングすることができない. よって, $\mathrm{OPT}_{P(T_{\mathrm{lp}})} = 0$ となる. 一方, 全ての木 T_{all} の中で図 3.1(b) の木だけはグラフ上に 1 回パッキングでき, $\mathrm{OPT}_{P(T_{\mathrm{all}})} = 1$ である.

次に $k+2$ 個の頂点から構成される別の問題例を与える:

$$
\begin{aligned}
&V = \{r, \mu, 1, 2, \ldots, k\},\\
&E = \{(\mu, r)\} \cup \{(i, r), (i, \mu), (\mu, i)\} \ \ (\forall i \in \{1, \ldots, k\}),\\
&b(\mu) = b(1) = \cdots = b(k) = 1,\ b(r) = +\infty,\\
&t((\mu, r)) = 1/(k-1),\ h((\mu, r)) = 0,\\
&t((i, r)) = 1,\ h((i, r)) = 0 \ \ (\forall i \in \{1, \ldots, k\}),\\
&t((i, \mu)) = 0,\ h((i, \mu)) = 0 \ \ (\forall i \in \{1, \ldots, k\}),\\
&t((\mu, i)) = (1+\varepsilon)/k,\ h((\mu, i)) = \varepsilon \ \ (\forall i \in \{1, \ldots, k\}).
\end{aligned}
$$

ここで, ε は $0 < \varepsilon \ll 1/k$ を満たす任意の値である. この問題例に対して線形緩和問題 $LP(T_{\mathrm{lp}})$ の最適解 $(\bar{x}_j \mid j \in T_{\mathrm{lp}})$ は一意的に定まり, $|T_{\mathrm{lp}}| = k$ および $\mathrm{OPT}_{LP(T_{\mathrm{lp}})} = k/(1+\varepsilon)$ を満たす (図 3.2(a) 参照). 元問題 $P(T_{\mathrm{all}})$ に対して任意の木 $j \in T_{\mathrm{lp}}$ はグラフ上にパッキングすることができないので, $\mathrm{OPT}_{P(T_{\mathrm{lp}})} = 0$ を満たす. 一方, $\mathrm{OPT}_{P(T_{\mathrm{all}})} = k-1$ である. なぜなら, 図 3.2(b) の木が元問題 $P(T_{\mathrm{all}})$ に対して $k-1$ 回パッキングできるので $\mathrm{OPT}_{P(T_{\mathrm{all}})} \geq k-1$ であり, 頂点 μ から出る辺 $\delta^+(\mu)$ を木として使用できる最大回数は,

$$
\left\lfloor \frac{b(\mu)}{\min_{e \in \delta^+(\mu)} t(e)} \right\rfloor = \left\lfloor \frac{1}{\min\{1/(k-1), (1+\varepsilon)/k\}} \right\rfloor = k-1
$$

で抑えられるので $\mathrm{OPT}_{P(T_{\mathrm{all}})} \leq k-1$ である. すなわち, この問題例では任意の頂点数 $|V|$ に対して $\mathrm{OPT}_{P(T_{\mathrm{lp}})} = 0$ かつ $\mathrm{OPT}_{P(T_{\mathrm{all}})} = |V| - 3$ が成り立つ. この問題例の最適解 $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ を任意に大きくすることができるので, SimpleLPBaseAL アルゴリズムの近似精度はいくらでも悪くなる.

以上より, $\mathrm{OPT}_{P(T_{\mathrm{all}})} \leq |V| - 3$ を満たす問題例の中に SimpleLPBaseAL アルゴリズムの近似精度がいくらでも悪くなってしまう問題例が存在する. 一方, $\mathrm{OPT}_{P(T_{\mathrm{all}})} \geq |V| + 1$ を満たす問題例に対しては定理 6 を満たす α を取ることがで, また, $\mathrm{OPT}_{P(T_{\mathrm{all}})} = |V|$ を満たす問題例については補題 1 よりアルゴリズムの出力した解の目的関数値が 1 以上 (すなわち, $\sum_{j \in T_{\mathrm{lp}}} \lfloor \bar{x}_j \rfloor \geq 1$) である保証が得られる. なお, $\mathrm{OPT}_{P(T_{\mathrm{all}})} = |V| - 1, |V| - 2$ については結果が得られていない.

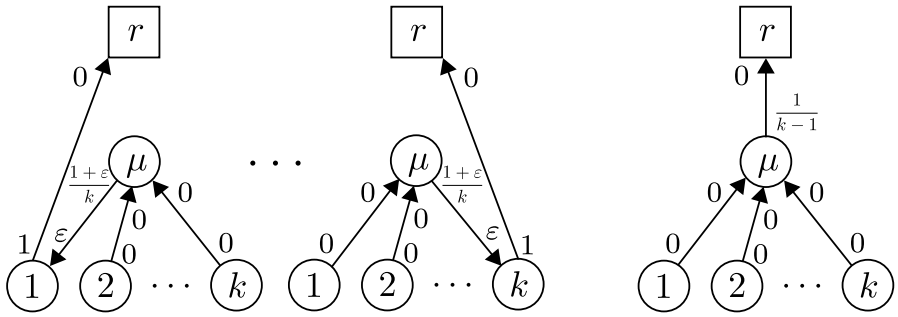


図 3.2: SimpleLPBaseAL アルゴリズムの近似精度がいくらでも悪くなってしまう問題例 ($|V| = k + 2$, $\mathrm{OPT}_{P(T_{\mathrm{all}})} = k - 1$, $\mathrm{OPT}_{P(T_{\mathrm{lp}})} = 0$)

3.5　結論

本章では, 頂点容量制約付き有向全域木パッキング問題に対する線形緩和に基づくアルゴリズムの近似精度について議論した. まず, 線形緩和に基づくアルゴリズムを一般化した SimpleLPBaseAL アルゴリズムを定義した. その後に, 最適値 $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ が頂点数 $|V|$ の α 倍 ($\alpha > 1$) である問題例に対して, SimpleLPBaseAL アルゴリズムが $(1 - 1/\alpha)$ 近似アルゴリズムであることを示した. また, 第 2 章で提案したアルゴリズも, 条件式 (2.5) で列生成法が停止したときは $(1-1/\alpha)$ 近似アルゴリズムであることを示した. 最後に, 任意の頂点数 $|V|$ に対して $\mathrm{OPT}_{P(T_{\mathrm{all}})} \leq |V| - 3$ を満たす問題例の中に SimpleLPBaseAL アルゴリズムの近似精度がいくらでも悪くなってしまう問題例が存在することを示した.

第4章　ラグランジュ緩和に基づくアルゴリズム

4.1　序論

本章では, 頂点容量制約付き有向全域木パッキング問題に対してラグランジュ緩和に基づくアルゴリズムを提案する. ラグランジュ緩和とは, 整数計画問題から一部の制約式を取り除き, それらの制約式を違反した量にラグランジュ乗数を乗じたものを目的関数に加える緩和法であり, ラグランジュ緩和した問題をラグランジュ緩和問題と呼ぶ. 線形緩和問題と同様に, ラグランジュ緩和問題は元の整数計画問題に比べて解きやすい問題となる.

本章で提案するアルゴリズムの基本的な流れは第 2 章で提案したアルゴリズムと同様であり, 2 段階に分けて問題を解く. 1 段階目ではパッキングする候補の木を生成する. ラグランジュ緩和問題に対して列生成法を適用し, 停止条件を満たすまで新たな木を生成および追加することを繰り返す. このとき, ラグランジュ緩和問題に対する良いラグランジュ乗数を得るために劣勾配法を適用する. 2 段階目では 1 段階目で生成した木の候補に対して, パッキングを行う回数を決定する. 第 2 章で提案したアルゴリズムよって得られた実行可能解の質が非常に良いことから, 同様の方法でパッキング回数を決定する.

以下では, 4.2 節でラグランジュ緩和の詳細, 4.3 で 1 段階目のアルゴリズム, 4.4 節で 2 段階目のアルゴリズムについて述べる. 提案アルゴリズムの全体像が分かるように 4.5 節でアルゴリズム全体の流れを示す. 次に, 4.6 節で提案アルゴリズムの計算実験の結果を示す. センサーネットワークに関する既存研究で使用されていたセンサー位置情報から生成した問題例と, ランダムに生成した問題例の 2 つに対して実験を行ったところ, 本章で提案したアルゴリズムは, 第 2 章で提案したアルゴリズムと比べて, 同じ数の木を生成するのに要する計算時間が短いにも関わらず, より良

い解を出力できることを示した. また, 一部の問題例に対しては厳密な最適解を得ることができた. 最後に, 4.7 節で本章の結論を述べる.

4.2　ラグランジュ緩和

1.5 節で述べたように, 整数計画問題である $P(T)$ を厳密に解くことは容易ではない. そのため, 第 2 章では $P(T)$ の線形緩和問題から得られる情報を利用した発見的解法を提案した. 本章のアルゴリズムでは, 線形緩和の代わりに, ラグランジュ緩和という別の代表的な緩和法を使用する. ラグランジュ緩和とは, 整数計画問題から一部の制約式を取り除き, それらの制約式を違反した量にラグランジュ乗数を乗じたものを目的関数に加える緩和法であり, ラグランジュ緩和した問題をラグランジュ緩和問題と呼ぶ. 線形緩和問題と同様に, ラグランジュ緩和問題は元の整数計画問題に比べて解きやすい問題となる. ラグランジュ問題を解くことにより, 元の整数計画問題対する有益な情報を得ることができ, この情報を利用した様々な発見的解法が提案されている. ラグランジュ緩和が線形緩和に対して優れている点は, 劣勾配法という良いラグランジュ乗数を定めるための発見的解法が存在することである. この手法は実装が簡単でカスタマイズし易い. しかし, パラメータが増えて設計するのが難しくなる側面もある.

$P(T)$ に対するラグランジュ問題 $LR(T,\lambda)$ は次のようになる:

$$
\begin{aligned}
LR(T,\lambda) \qquad \text{maximize} \quad & \sum_{j\in T} x_j + \sum_{i\in V} \lambda_i \left(b_i - \sum_{j\in T} a_{ij} x_j \right) && (4.1)\\
& = \sum_{j\in T} c_j(\lambda) x_j + \sum_{i\in V} \lambda_i b_i, && (4.2)\\
\text{subject to} \quad & x_j \in \{0,1,\ldots,u_j\}, \qquad && \forall j \in T.
\end{aligned}
$$

ここで, $\lambda_i \geq 0$ は頂点 $i \in V$ に対するラグランジュ乗数であり, u_j は木 $j \in T$ のみをパッキングしたときの最大パッキング回数である (すなわち, $u_j = \min_{i\in V:\, a_{ij}>0} \lfloor b_i/a_{ij} \rfloor$). また, $\lambda = (\lambda_i \mid i \in V)$ はラグランジュ乗数ベクトルであり (以降は混乱のない限り単にラグランジュ乗数と呼ぶ), 各木 $j \in T$ 対する $c_j(\lambda) = 1 - \sum_{i\in V} a_{ij}\lambda_i$ は相対コストである. 相対コスト $c_j(\lambda)$ は直感的には木 j の価値を表す関数と捉えることができ

る．さらに, ラグランジュ緩和問題 $LR(T, \lambda)$ の最適解を $\mathrm{OPT}_{LR(T,\lambda)}$ とし, 最適解を $x(\lambda)$ と表すことにする．

任意の木集合 $T \subseteq T_{\mathrm{all}}$ に対して, 任意のラグランジュ乗数 $\lambda \geq 0$ が与えられたとき, ラグランジュ緩和問題 $LR(T, \lambda)$ の最適解 $x(\lambda) = (x_j(\lambda) \mid j \in T)$ は次のように簡単に計算することができる:

$$x_j(\lambda) = \begin{cases} u_j, & (c_j(\lambda) > 0), \\ 0, & (c_j(\lambda) \leq 0), \end{cases} \quad \forall j \in T. \tag{4.3}$$

一般的に, 任意のラグランジュ乗数 $\lambda \geq 0$ に対して, ラグランジュ緩和問題の最適値は元問題の最適値の上界を与える．すなわち, $\mathrm{OPT}_{P(T)}$ が整数値であることに注意すると, 任意の $\lambda \geq 0$ に対して $\lfloor \mathrm{OPT}_{LR(T,\lambda)} \rfloor$ は $\mathrm{OPT}_{P(T)}$ の上界となる．ただし, $T \neq T_{\mathrm{all}}$ のとき, $\mathrm{OPT}_{LR(T)}$ は $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ の上界になるとは限らない．

4.3 木生成アルゴリズム

本節では, 候補となる木の集合 T を生成する 1 段階目のアルゴリズム LRGenInTrees を提案する．このアルゴリズムの大まかな流れは 2.3 節で提案した GenInTrees アルゴリズムと同じである．まず，提案アルゴリズムは 4.3.1 節で述べる簡単なアルゴリズムによって初期木集合を生成する．次に, 現在の木集合 T に対して良いラグランジュ乗数 λ を得るために劣勾配法と呼ばれる発見的解法を適用する．その後，得られた λ に対するラグランジュ緩和問題 $LR(T, \lambda)$ の最適解の情報を用いて新たな木を生成し，現在の木集合 T に追加することを繰り返し行う．この手法は一般的に列生成法と呼ばれるもので, 各反復で生成される木は現在の木集合 T に追加することによって最適値 $\mathrm{OPT}_{LR(T)}$ を改善するものである．列生成法によって生成された木集合は元問題 $P(T_{\mathrm{all}})$ に対して良い解を得るために有効であると考えられる．以下では, 4.3.2 節において劣勾配法の詳細について, 4.3.3 節において列生成法の詳細について，4.3.4 節において列生成法の停止条件について述べたのち, LRGenInTrees の全体像について 4.3.5 節にまとめる．また, 劣勾配法の高速化手法について 4.3.6 節で詳しく述べる．

4.3.1　初期木集合

本章で提案するラグランジュ緩和に基づくアルゴリズムは, 木集合を生成するために第 2 章で提案した線形緩和に基づくアルゴリズムと同じく列生成法を用いる. そのため, 1 本の初期木さえあれば実行できる. しかしながら, 予備実験により予め数本の木をまとめて初期木集合として与えたほうが, 計算時間が短くなることが線形緩和に基づくアルゴリズムと同様に観測された. また, ランダムに生成された初期木の本数を $|V|$ 本より多く増やしても, 計算時間はそれほど減少しないことも観測した. 以上の考察より, ラグランジュ緩和に基づくアルゴリズムにおいても, 初期木集合として $|V|$ 本のランダムに生成された木を用いることにした. 具体的な初期木集合の生成方法は 2.3.1 節と同様である.

4.3.2　劣勾配法

4.2 節で示したように, 任意のラグランジュ乗数 $\lambda \geq 0$ に対してラグランジュ緩和問題の最適解 $x(\lambda)$ を求めることは非常に簡単である. しかし, 一般的に最適なラグランジュ乗数 λ を求めることは簡単でない. そこで本章では, 現在の木集合 T に対して, 良いラグランジュ乗数 λ を得るために劣勾配法と呼ばれる発見的解法を使用する. 劣勾配法は, 良いラグランジュ乗数を得るためのアルゴリズムとして, よく知られた手法である [1, 10, 18]. 良いラグランジュ乗数 λ を得ることは $P(T)$ に対する良い上界 ($\mathrm{OPT}_{LR(T,\lambda)}$) を得るために必要である.

ラグランジュ乗数 λ に関する劣勾配 $s(\lambda) = (s_i \mid i \in V)$ を, 各頂点 $i \in V$ に対して $s_i(\lambda) = b_i - \sum_{j \in T} a_{ij} x_j(\lambda)$ と定義する. 劣勾配法はこの劣勾配 $s(\lambda)$ に従って, 次のような更新式で繰り返しラグランジュ乗数ベクトル λ を初期ベクトルから更新する手法である:

$$\lambda_i := \max\left(0, \lambda_i - \pi \frac{\mathrm{UB}(\lambda) - \mathrm{LB}}{\sum_{i \in V}\{s_i(\lambda)\}^2} s_i(\lambda)\right), \quad \forall i \in V. \tag{4.4}$$

ここで, $\mathrm{UB}(\lambda) = \mathrm{OPT}_{LR(T,\lambda)}$ は $P(T)$ の上界, LB は $P(T)$ の下界 (実行可能解の目的関数値), $\pi \geq 0$ はステップサイズを調整するパラメータである. $\theta(\lambda) = \pi(\mathrm{UB}(\lambda) - \mathrm{LB})/(\sum_{i \in V}\{s_i(\lambda)\}^2)$ と定義する. $\theta(\lambda)$ は一般的にステップサイズと呼ばれる.

劣勾配法の具体的な繰り返し手順について説明する. 初めに, $P(T)$ の下界 LB, 初期ラグランジュ乗数 λ, およびパラメータ π の初期値が劣勾配法に与えれる. 次に,

式 (4.4) によって N 回ラグランジュ乗数 λ の更新を行う．N 回の反復の間に最良の最適値 $\mathrm{OPT}_{LR(T,\lambda)}$ が更新されなかったときはパラメータ π の値を半分にする．もしパラメータ π の値が π_{end} よりも小さくなったら, 劣勾配法の反復を止める．そうでなければ, 再び N 回ラグランジュ乗数 λ の更新を行う．本章の計算実験では $N = 30$, $\pi_{\mathrm{end}} = 0.005$ と設定した．これらのパラメータ値は劣勾配法で一般的に用いられる値である [2].

基本的な劣勾配法の流れは上記の通りであるが, 本研究の劣勾配法ではラグランジュ乗数 λ の更新方法を僅かに修正した．式 (4.4) において, 劣勾配 $s_i(\lambda)$ の代わりに修正を加えた劣勾配 $s'_i(\lambda)$ を使用する．各頂点 $i \in V$ に対して, もし $\lambda_i = 0$ と $s_i(\lambda) < 0$ を満たすならば, $s'_i(\lambda) = 0$ と修正し, そうでなければ, $s'_i(\lambda) = s_i(\lambda)$ とする．$\lambda_i = 0$ と $s_i(\lambda) < 0$ を満たすようなラグランジュ乗数 λ_i は, 式 (4.4) で更新されないことが予め分かっているので, $s'_i(\lambda) = 0$ とすることで, 本来更新に寄与しない要素の影響を避けることができる．

SubOpt$(T, \mathrm{LB}, \lambda, \pi)$ を本研究で使用する劣勾配法と定義する．木集合 T に対する下界 LB, 初期ラグランジュ乗数 λ, およびパラメータ π の初期値が SubOpt の入力である．SubOpt はラグランジュ乗数 λ とパラメータ π の ϱ 個のペア $(\lambda^{(1)}, \pi^{(1)}), \ldots, (\lambda^{(\varrho)}, \pi^{(\varrho)})$ を出力する．各 $k = 1, \ldots, \varrho$ に対して, ラグランジュ乗数 $\lambda^{(k)}$ は, 劣勾配法の各反復で生成されたラグランジュ乗数 λ の中で, $\mathrm{OPT}_{LR(T,\lambda)}$ が小さい方から k 番目の値を達成するものであり, $\pi^{(k)}$ は $\lambda^{(k)}$ を生成した時のパラメータ π の値である．ここで, ϱ は SubOpt が出力するペアの数を定めるパラメータである．これらの出力されたペアは次節で述べる列生成法の中で新たな木を生成するのに使用される．SubOpt の詳細を擬似コードで示す．擬似コードにおいて, $\mathrm{UB}_{\mathrm{best}}$ は劣勾配法の反復の中で得られた最良の (最小の) 上界 (最適値 $\mathrm{OPT}_{LR(T,\lambda)}$) を表す．

次に, SubOpt に与える初期入力について述べる．もし SubOpt が 4.3.1 節で述べた初期木集合に適用される場合は, 全ての頂点 $i \in V$ に対して初期ラグランジュ乗数を $\lambda_i = 1/\min_{j \in T} \sum_{v \in V} a_{vj}$ とし, ステップサイズの調整パラメータの初期値を $\pi = 2$ とした．これらの簡単な初期入力値は SubOpt の最初の呼び出しだけに使用され, 提案アルゴリズム全体の性能には大きく影響しない．2 回目以降の呼び出し (すなわち, 列生成法によって新たな木が追加された木集合 T に対して SubOpt が適用される場合) では, 提案アルゴリズムは SubOpt の最後の実行における情報を使用する．

Algorithm SubOpt$(T, \mathrm{LB}, \lambda, \pi)$

Input: 木集合 T, $P(T)$ の下界 LB, 初期ラグランジュ乗数 λ, パラメータ π の初期値.

Output: ϱ 個のペア $(\lambda^{(1)}, \pi^{(1)}), \ldots, (\lambda^{(\varrho)}, \pi^{(\varrho)})$. ここで, 各 $k = 1, \ldots, \varrho$ に対して, ラグランジュ乗数 $\lambda^{(k)}$ は生成されたラグランジュ乗数の中で $\mathrm{OPT}_{LR(T,\lambda)}$ が小さい方から k 番目の値を達成するものであり, $\pi^{(k)}$ は $\lambda^{(k)}$ を生成した時のパラメータ π の値である.

1: $\mathrm{UB}_{\mathrm{best}} := +\infty$ および $\Lambda := \emptyset$ とする.
2: 3 から 10 までを N 回繰り返す.
3: ラグランジュ緩和問題 $LR(T, \lambda)$ の最適値 $\mathrm{OPT}_{LR(T,\lambda)}$ と最適解 $x(\lambda)$ を計算し, $\mathrm{UB} := \mathrm{OPT}_{LR(T,\lambda)}$ とする.
4: もし $|\Lambda| < \varrho$ ならば, $\Lambda := \Lambda \cap (\lambda, \pi)$ とし, 6 に行く.
5: もし $(\lambda^{(\varrho)}, \pi^{(\varrho)}) \in \Lambda$ よって得られた ϱ 番目の上界 (最適値 $\mathrm{OPT}_{LR(T,\lambda)}$) よりも小さければ, $\Lambda := \Lambda \setminus (\lambda^{(\varrho)}, \pi^{(\varrho)}) \cap (\lambda, \pi)$ とする.
6: もし $\mathrm{UB}_{\mathrm{best}} > \mathrm{UB}$ ならば, $\mathrm{UB}_{\mathrm{best}} := \mathrm{UB}$ と更新する.
7: 全ての頂点 $i \in V$ に対して, 劣勾配 $s_i := b_i - \sum_{j \in T} a_{ij} x_j(\lambda)$ を計算する.
8: 各頂点 $i \in V$ において, もし $\lambda_i = 0$ と $s_i < 0$ を満たすならば, $s_i := 0$ と修正する.
9: ステップサイズ $\theta := \pi(\mathrm{UB} - \mathrm{LB})/(\sum_{i \in V} s_i^2)$ を計算する.
10: 全ての頂点 $i \in V$ に対して, $\lambda_i := \max(0, \lambda_i - \theta s_i)$ によってラグランジュ乗数を更新する.
11: もし N の反復の間に $\mathrm{UB}_{\mathrm{best}}$ が更新されなかったら, $\pi := \pi/2$ とする.
12: もし $\pi < 0.005$ ならば, ϱ 個のペア $(\lambda^{(1)}, \pi^{(1)}), \ldots, (\lambda^{(\varrho)}, \pi^{(\varrho)}) \in \Lambda$ を出力し, 停止する.
13: 2 に戻る.

具体的には, 列生成法において最後の新たな木を生成するために使用したラグランジュ乗数を $\lambda^{(k')}$ (すなわち, それに対応するペアは $(\lambda^{(k')}, \pi^{(k')})$) とすると, SubOpt に入力する初期値を $\lambda := \lambda^{(k')}$ および $\pi := 4\pi^{(k')}$ とする. この手法により, 良いラグランジュ乗数を得るために必要な SubOpt の反復回数を減らすことができる. $\pi^{(k')}$ の値を 4 倍して π の初期値に用いる理由について簡単に説明する. $\pi^{(k')}$ は良い最適値 $\mathrm{OPT}_{LR(T,\lambda)}$ を得たときに使用していた値なので, 0.005 に非常に近い値である可能性が高い. そのため, $\pi := \pi^{(k')}$ としてしまうと, ラグランジュ乗数 λ の十分な探索が行われないまま終了してしまう可能性が高い. また, ステップサイズ $\theta(\lambda)$ が小さくなりすぎてラグランジュ乗数 λ の更新量が足りず, 逆に SubOpt の反復回数が増大する危険性がある. 実際に, 予備実験において $\lambda^{(k')}$ と近いラグランジュ乗数が多く生成される傾向にあった. よって, もう少し大きい初期値をパラメータ π に与えなければならない. 予備実験により, $\pi = 4\pi^{(k')}$ とすれば, $\lambda^{(k')}$ と近いラグランジュ乗数が多く生成されるという傾向がなくなり, SubOpt でラグランジュ乗数 λ をうまく探索できることを観測した.

$P(T)$ の実行可能解を生成するために 2.4 節で貪欲アルゴリズム PackInTrees を提案した. このアルゴリズムは各木の評価基準として現在の残り容量に対する最大パッキング回数を用いた構築型のアルゴリズムであった. 全ての木 $j \in T$ に対して $x_j^{(0)} := 0$ である解を初期解として PackInTrees アルゴリズムを実行し, そのとき出力された $P(T)$ の実行可能解の目的関数値を LB とした. また, 提案アルゴリズムは SubOpt を呼び出す度に LB を更新しない (PackInTrees アルゴリズムを呼び出さない). 実際には, 提案アルゴリズムでは新たな木を 100 本追加する度に PackInTrees を呼び出す. このようにすると SubOpt の反復中には必ずしも良い $P(T)$ の実行可能解 (すなわち, 良い下界) が得られないが, 予備実験により提案アルゴリズムの性能は下界の質に大きく影響を受けないことを確認している.

上記の提案アルゴリズムの説明は基本部分についてだけ述べている. 実際には上記のアルゴリズムに高速化手法を組み込み, SubOpt で使用する木の数を減らしたり, 各反復における実際の計算量を削減したりしている. これらの詳細については 4.3.6 節で述べる.

なお, 一般性を失うことなく, 全ての頂点 $i \in V$ に対して $b_i = b$ (b は任意の正の定数) と仮定することができる. 例えば, 全ての辺 $(v, w) \in E$ に対して, $h((v, w)) :=$

$h((v,w))/b_w$, $t((v,w)) := t((v,w))/b_v$ としたのち, 全ての頂点 $i \in V$ に対して $b_i := 1$ と修正することにより, 頂点容量を正規化することができるからである. このような正規化は劣勾配法の実行を安定化させるのに効果があることが知られているため, 提案アルゴリズムにおいてもこのような正規化を行った上でアルゴリズムを適用している.

4.3.3　列生成法

本章で提案するラグランジュ緩和に基づくアルゴリズムも, 第 2 章で提案した線形緩和に基づくアルゴリズムと同じく, 木集合を生成するために列生成法を用いる. 基本的な流れは 2.3.3 節で示した列生成法と同様であり, 列生成法は 4.3.1 節で得られた初期木集合 $T \subseteq T_{\mathrm{all}}$ から始め，停止条件を満たすまで繰り返し新たな木を生成し，木集合 T に追加する手法である．各繰り返しで生成される木は，現在の木集合 T に追加することによって最適値 $\mathrm{OPT}_{LR(T,\lambda)}$ が改善する.

どのような木を現在の木集合 T に追加するかについて具体的に議論する. 任意のラグランジュ乗数 λ に対して正の相対コスト ($c_j(\lambda) > 0$) を持つ全ての木の集合を $T^+(\lambda) \subseteq T_{\mathrm{all}}$ とする (すなわち, $T^+(\lambda) = \{j \in T_{\mathrm{all}} \mid c_j(\lambda) > 0\}$). 任意のラグランジュ乗数 λ に対する最適解 $\mathrm{OPT}_{LR(T,\lambda)}$ の求め方 (4.2 節を参照) より, もし現在の木集合 T が $T^+(\lambda) \subseteq T$ を満たすならば, $LR(T,\lambda)$ の最適解 $\mathrm{OPT}_{LR(T,\lambda)}$ は $LR(T_{\mathrm{all}},\lambda)$ の最適解 $\mathrm{OPT}_{LR(T_{\mathrm{all}},\lambda)}$ と等しい. 言い換えると, もし現在の木集合 T に含まれていないような木 $\tau \in T_{\mathrm{all}} \setminus T$ の中に, 正の相対コスト ($c_\tau(\lambda) > 0$) を持つ木が存在するならば, $LR(T,\lambda)$ の最適解 $x(\lambda)$ は $LR(T_{\mathrm{all}},\lambda)$ に対して最適でない. よって,

$$c_\tau(\lambda) = 1 - \sum_{i \in V} a_{i\tau}\lambda_i > 0.$$

すなわち,

$$\sum_{i \in V} a_{i\tau}\lambda_i < 1 \tag{4.5}$$

を満たす新たな木 $\tau \in T_{\mathrm{all}} \setminus T$ を現在の木集合 T に追加することによって, 最適値 $\mathrm{OPT}_{LR(T,\lambda)}$ が改善する．一般的に, このような条件を満たす木を探す問題は価格付け問題と呼ばれる.

式 (4.5) は, 全ての頂点 $i \in V$ に対して $y_i^* := \lambda_i$ としたとき, 2.3.2 節で示した式 (2.1) と等しい. よって, 木の探索範囲を $T_{\mathrm{all}} \setminus T$ から全ての木 T_{all} に変更し, 各辺 $(v, w) \in E$ に対して式 (2.2) と同様にコスト $\phi((v, w))$ を与えると, 式 (2.4) のように変形することができる. すなわち, 線形緩和に基づくアルゴリズムと同じく式 (4.5) の価格付け問題は最小重み根指定有向全域木問題に変換して解くことができる.

しかしながら, 与えられるラグランジュ乗数 λ は双対問題 $D(T)$ の最適解 y^* のように良い特徴 (現在の木集合に含まれている全ての木 $j \in T$ に対して, 双対問題 $D(T)$ の制約式 $\sum_{i \in V} a_{ij} y_i^* \geq 1$ を満たしているという性質) を持っていない. すなわち, 相対コストが必ずしも $c_j(\lambda) \leq 0$ を満たしているとは限らないので, 既に現在の木集合 T に含まれている木を生成するかもしれない. つまり, 2.3.2 節で示した最小重み根指定有向全域木問題に変換して解く方法では厳密には価格付け問題を解いていない. しかし, 予備実験より, (SubOpt によって得られてた) 十分に精度の良いラグランジュ乗数 λ を与えれば, 重複した木が生成されることは稀であることを観測した.

現在の木集合 T に含まれていない木を生成する確率を高めるために, 提案アルゴリズムでは 1 つだけではなく複数のラグランジュ乗数ベクトル λ 対して価格付け問題を解く方法を用いた. SubOpt では, 最良の上界を達成する ϱ 個のラグランジュ乗数ベクトルを出力し, 列生成法において新たな木 $\tau \in T_{\mathrm{all}} \setminus T$ が生成されるか, 全てのラグランジュ乗数ベクトル $\lambda^{(1)}, \ldots, \lambda^{(\varrho)}$ を調べるまで, $k = 1$ から順番に $\lambda^{(k)}$ に対する価格付け問題を解く. もし新たな木 τ が生成されたときは, 現在の木集合 T に加える. そうでないときは, 十分な量の木が生成できたと判断して列生成法を停止する.

4.3.4 停止条件

本節では列生成法の停止条件について議論する. 2.3.3 節で示した線形緩和に基づくアルゴリズムの停止条件と同様のものは使用できない. なぜなら, 2.3.3 節の停止条件では, 任意の木集合 $T \subseteq T_{\mathrm{all}}$ に対して $\mathrm{OPT}_{LP(T)} \leq \mathrm{OPT}_{LP(T_{\mathrm{all}})}$ が常に成り立つという性質を用いたが, ラグランジュ緩和問題の最適値 $\mathrm{OPT}_{LR(T,\lambda)}$ にはそのような性質がないためである. すなわち, 任意の木集合 $T \subseteq T_{\mathrm{all}}$, および任意のラグランジュ乗数 $\lambda, \lambda' \geq 0$ に対して, $\mathrm{OPT}_{LR(T,\lambda)} \leq \mathrm{OPT}_{LR(T_{\mathrm{all}},\lambda')}$ が常に成立するとは限らない. そこで, ラグランジュ緩和に基づくアルゴリズムでは新たに 2 つの停止条件を用いた.

1 つ目は元問題 $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ の上界によるものである. 定理 5 より, 全ての頂点 $i \in V$

に対して $\hat{y}_i := \lambda_i$ とすれば, 列生成法の各反復で線形緩和に基づくアルゴリズムと同様に最適値 $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ の上界を得ることができる. すなわち, 列生成法の各反復で得られた SubOpt のラグランジュ乗数 λ に対して, $\rho_\lambda = \min_{j \in T_{\mathrm{all}}} \left\{ \sum_{i \in V} a_{ij} \lambda_i \right\}$ とし, 目的関数値 $\omega = \sum_{i \in V} b_i (\lambda_i / \rho_\lambda)$ を求めることで, 列生成法の各反復において最適値 $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ の上界を得ることができる. UB^* を今までの反復の中で最良の上界 (すなわち, 最小の上界) と定義する. 現在の木集合 T が十分に良い (有益な木が多く含まれている) ものでなければ, UB^* は頻繁に更新される. 一方, T が十分に良い木集合であれば UB^* の更新は頻繁に起きない. そこで, 提案アルゴリズムでは UB^* が $|V|$ 回連続して更新されないとき列生成法を停止する.

2つ目は4.3.3節で示したように生成された木の重複に関するものである. SubOpt によって得られた ϱ 個のラグランジュ乗数ベクトル全てに対して価格付け問題 (最小重み根指定有向全域木問題) を解いた際に, 現在の木集合 T に含まれない新たな木 $\tau \in T_{\mathrm{all}} \setminus T$ が生成されなかったとき, すなわち, 生成した ϱ 個の木が全て既に T に含まれているものであったとき, 列生成法を停止する.

パラメータ ϱ の値は極端に小さい値でなければアルゴリズムの性能に大きな影響を与えない. そこで, 4.6節の計算実験では $\varrho := 10$ とした. 実際, 4.6節の計算実験において, 提案アルゴリズムは2つ目の停止条件で列生成法が停止することはなかった.

4.3.5 提案アルゴリズム

本節ではラグランジュ緩和に基づく木集合を生成するアルゴリズムの全体像を示す. 本研究では, この木集合を生成するアルゴリズムを LRGenInTrees と呼ぶことにする. LRGenInTrees アルゴリズムを擬似コード形式で以下に示す.

4.3.6 劣勾配法の高速化手法

本研究では, 4.3.2節の劣勾配法に対して2つの高速化手法を提案する. 1つ目は劣勾配法で実際に使用する木の数を削減することである. 予備実験により, 初期木および列生成法の早い段階で生成された木のほとんどは, 劣勾配法を実行する上で必要でないことを観測した (4.6.2節を参照). もし, SubOpt の各反復で生成された全てのラグランジュ乗数 λ に対して $x_j(\lambda) = 0$ を満たすような木 $j \in T$ が存在するなら

Algorithm LRGenInTrees

Input: 有向グラフ $G = (V, E)$, 根 $r \in V$, 各辺上の始点側と終点側の消費量 $t : E \to \mathbb{R}_+$ と $h : E \to \mathbb{R}_+$, 各頂点 $i \in V$ の容量 $b_i \in \mathbb{R}_+$, パラメータ ϱ.

Output: 木集合 T.

1: ランダムに生成した $|V|$ 本の初期木を木集合 T とし, 全ての頂点 $i \in V$ に対して初期ラグランジュ乗数を $\lambda_i := 1/\min_{j \in T} \sum_{v \in V} a_{vj}$ とする. また, $\mathrm{UB}^* := +\infty$, $\xi := 0$, $\pi := 2$ とする.

2: PackInTrees を呼び出し, 得られた実行可能解の目的関数値 (下界) を LB とする.

3: SubOpt$(T, \mathrm{LB}, \lambda, \pi)$ を呼び出し, ϱ 個のペア $(\lambda^{(1)}, \pi^{(1)}), \ldots, (\lambda^{(\varrho)}, \pi^{(\varrho)})$ を得る. また, $\xi := \xi + 1$ と更新する.

4: **for** $k = 1$ **to** ϱ **do**

5: グラフ G 上の全ての辺 $(v, w) \in E$ に対して, 辺コスト $\phi((v, w)) := \lambda_v^{(k)} t((v, w)) + \lambda_w^{(k)} h((v, w))$ を定義する.

6: 辺コスト ϕ に対して Edmonds のアルゴリズムを実行する. そのとき得られた最小合計コストを持つ木を τ, その合計コストを ρ とする.

7: もし $\sum_{i \in V} b_i(\lambda_i/\rho) < \mathrm{UB}^*$ を満たすならば, $\mathrm{UB}^* := \sum_{i \in V} b_i(\lambda_i/\rho)$ および $\xi := 0$ と更新する.

8: もし $\tau \notin T$ を満たすならば, $T := T \cap \{\tau\}$, $\lambda := \lambda^{(k)}$ および $\pi := 4\pi^{(k)}$ とし, 11 に進む.

9: **end for**

10: 木集合 T を出力して停止する.

11: もし $\xi = |V|$ を満たすならば, 10 に戻る.

12: もし最後の PackInTrees の呼び出しから T に新たな木を 100 本追加したのならば, PackInTrees を呼び出し, LB を更新する.

13: 3 に戻る.

ば, そのような木 j を T から取り除いたとしても SubOpt の動作に全く影響を与えない. この考察から, そのような不必要な木を SubOpt に使用しない仕組みを導入する. しかしながら, SubOpt の実行前に不必要な木かどうかを判断するのは困難である. そこで, SubOpt の過去の動作に基づいて, 必要な木であるか否かを推定する方法を用いた.

提案アルゴリズムは SubOpt の最近の数回の呼び出しで使用頻度が小さい木を不必要と判断し, 以降の SubOpt の最近の呼び出しでは使用しない. 具体的には, LRGenInTrees アルゴリズムの各反復において SubOpt(T, LB, λ, π) を呼び出す代わりに SubOpt(T', LB, λ, π) を呼び出す. ここで, T' は次のような規則で T から不必要と判断された木を取り除いて得られた木集合である: SubOpt の呼び出しが完了する度に, 最後の SubOpt の実行の間に最適解 $\mathrm{OPT}_{LR(T,\lambda)}$ に使用された (すなわち, $x_j(\lambda) > 0$ である) 回数が β より小さい全ての木に対して, “おそらく不必要” とラベル付けする (β は “おそらく不必要” とラベル付けする木の数を調整するパラメータである). SubOpt の呼び出し後に γ 回連続で “おそらく不必要” と判断された全ての木を T から取り除いくとによって得られた木集合を T' する (γ は不必要と判断されるまでに必要な SubOpt の呼び出し回数を定めるパラメータである). β をなるべく大きな値に, γ をなるべく小さな値にすることで, SubOpt で使用する木の数を多く削減することができる. しかし, あまりに極端な値を設定すると SubOpt の動作に必要な木まで削除されてしまい, LRGenInTrees アルゴリズム全体の挙動に大きな影響を与えるので注意が必要である. 本研究では, 予備実験により十分安全に木を削減できる値を調べ, $\beta = 5$ および $\gamma = |V|$ とした.

2 つ目の高速化手法は劣勾配法の各反復の実際の計算量を削減することである. SubOpt の 1 回の反復は擬似コードの 3 行目から 10 行目までの部分に対応する. SubOpt を擬似コードで示した通り素直に実装したとすると, 3 行目と 7 行目は $\Theta(|V||T|)$ 時間かかり, ボトルネックとなる (他の行は $O(|V|)$ 時間で実行できる). 以下では, この SubOpt の 3 行目と 7 行目の高速化について議論する.

3 行目では, ラグランジュ緩和問題 $LR(T, \lambda)$ の最適値 $\mathrm{OPT}_{LR(T,\lambda)}$ と最適解 $x(\lambda)$ が計算される. 初めに, 最適解 $x(\lambda)$ を計算する方法について議論する. 式 (4.3) を思い出すと, ラグランジュ乗数 λ が更新された後に最初から最適解 $x(\lambda)$ を求めるには, 全ての木 $j \in T$ に対して相対コスト $c_j(\lambda)$ が決定されている必要がある. $c_j(\lambda)$

の計算は各木 $j \in T$ に対して $O(|V|)$ 時間かかるので, 単純に実装すると最適解 $x(\lambda)$ の計算には $\Theta(|V||T|)$ 時間が必要である. ここで, 前回の反復のラグランジュ乗数 λ' に対する最適解 $x_j(\lambda')$ を保持していると仮定すると, 相対コストが $c_j(\lambda') > 0$ から $c_j(\lambda) \leq 0$ に変化した, またはその逆の変化をした木 j に対してだけ $x(\lambda')$ を更新すれば, 現在のラグランジュ乗数 λ の最適解 $x(\lambda)$ として使用可能であることが分かる. さらに, (SubOpt の初期の反復を除いて) 各反復におけるラグランジュ乗数 λ の変化量は小さので, ほとんどの木 j において $x_j(\lambda')$ から値が変化しない (すなわち, $x_j(\lambda) = x_j(\lambda')$ である) 傾向にある. これらの考察により, 相対コスト $c_j(\lambda)$ の上界と下界を導入し, 上界と下界から $x_j(\lambda)$ の値を簡単に決定できるような木 j に対して, $c_j(\lambda)$ の真の値を計算しなくても済む方法を提案する.

SubOpt の各反復において, 全ての木 $j \in T$ に対して相対コスト $c_j(\lambda)$ の上界 $c_j^{\mathrm{UB}}(\lambda)$ と下界 $c_j^{\mathrm{LB}}(\lambda)$ を保持していると仮定する. すると, 相対コスト $c_j(\lambda)$ の真の値を知らなくても, 各木 $j \in T$ に対して, $c_j^{\mathrm{UB}}(\lambda) \leq 0$ のとき $x_j(\lambda) = 0$, $c_j^{\mathrm{LB}}(\lambda) > 0$ のとき $x_j(\lambda) = u_j$ であるとすぐに決定することができる. そのどちらでもない場合にのみ (すなわち, $c_j^{\mathrm{UB}}(\lambda) > 0$ かつ $c_j^{\mathrm{LB}}(\lambda) \leq 0$ であるとき), $c_j(\lambda)$ の真の値を計算すればよい. 高速に計算できる質の良い上界 $c_j^{\mathrm{UB}}(\lambda)$ と下界 $c_j^{\mathrm{LB}}(\lambda)$ を与えることができれば, 実際の計算量を削減することができる. しかしながら, 最悪の計算量は $O(|V||T|)$ から変わることはない. どのような上界 $c_j^{\mathrm{UB}}(\lambda)$ と下界 $c_j^{\mathrm{LB}}(\lambda)$ を保持するかについて議論する前に, 相対コスト $c_j(\lambda)$ の計算の省略が SubOpt の他の部分に影響を与えないことを確認する.

この影響があるのは, 最適値 $\mathrm{OPT}_{LR(T,\lambda)}$ を計算する部分だけである. 最適解 $x(\lambda)$ を求めるときに, いくつかの木に対して相対コスト $c_j(\lambda)$ の真の値の計算を省略した. そのため, 式 (4.2) によって $\mathrm{OPT}_{LR(T,\lambda)}$ を計算することは不可能である. そこで, 代わりに式 (4.1) によって $\mathrm{OPT}_{LR(T,\lambda)}$ を計算する. 式 (4.1) は次のように変形できる:

$$\mathrm{OPT}_{LR(T,\lambda)} = \sum_{j \in T} x_j(\lambda) + \sum_{i \in V} \lambda_i \left(b_i - \sum_{j \in T} a_{ij} x_j(\lambda) \right) = \sum_{j \in T} x_j(\lambda) + \sum_{i \in V} \lambda_i s_i(\lambda).$$

よって, 劣勾配 $s(\lambda)$ が与えられていれば, $\mathrm{OPT}_{LR(T,\lambda)}$ を $O(|V| + |T|)$ 時間で計算することができる. SubOpt の擬似コードの中では, 劣勾配 $s(\lambda)$ は最適解 $\mathrm{OPT}_{LR(T,\lambda)}$ を計算した後に求められる. しかしながら, 最適解 $\mathrm{OPT}_{LR(T,\lambda)}$ を求める前に劣勾配 $s(\lambda)$ を計算しても問題のないことがすぐに分かる. すなわち, SubOpt の 3 行目と 7

行目の計算順序を次のように修正する: 初めに最適解 $x(\lambda)$ を計算する. 次に劣勾配 $s(\lambda)$ を計算し, 最後に最適値 $\mathrm{OPT}_{LR(T,\lambda)}$ を計算する.

次に7行目の劣勾配 $s(\lambda)$ の計算方法に焦点を当てる. 3行目と同様に, 単純に実装すると $\Theta(|V||T|)$ 時間かかる. 式 (4.3) より, 全ての木 $j \in T$ に対して $x_j(\lambda)$ は0と u_j のどちらかの値しか取らないことが分かる. そのため, $x_j(\lambda) = u_j$ である木 j しか劣勾配 $s(\lambda)$ の計算に寄与しない. よって, 劣勾配 $s(\lambda)$ の計算を次ようにする:

$$s_i(\lambda) = b_i - \sum_{\substack{j \in T \\ x_j(\lambda) = u_j}} a_{ij} x_j(\lambda).$$

また, 予備実験により, SubOpt の多くの反復において最適解の多くの変数が $x_j(\lambda) = 0$ であることを観測した. すなわち, 最悪の計算量は $O(|V||T|)$ から変わらないが, 実際の計算量を削減することができる.

ここからは, どのような相対コストの上界 $c_j^{\mathrm{UB}}(\lambda)$ と下界 $c_j^{\mathrm{LB}}(\lambda)$ を用いるかについて示す. SubOpt の最初の反復では, 相対コスト $c_j(\lambda)$ の真の値を計算し, 全ての木 $j \in T$ に対して $c_j^{\mathrm{UB}}(\lambda) := c_j(\lambda)$ および $c_j^{\mathrm{LB}}(\lambda) := c_j(\lambda)$ とする. 2回目以降の反復では $c_j^{\mathrm{UB}}(\lambda)$ と $c_j^{\mathrm{LB}}(\lambda)$ を前回の反復の情報を用いて更新する. 木 j における頂点 i の資源消費量 a_{ij} に対して, $a_i^{\max}$ および $a_i^{\min}$ を次のように定義する:

$$a_i^{\max} = \max_{j \in T} a_{ij}, \tag{4.6}$$

$$a_i^{\min} = \min_{j \in T} a_{ij}. \tag{4.7}$$

前回の反復で得られたラグランジュ乗数を λ' とし, 現在の反復で式 (4.4) によって λ' から λ に更新されと仮定する. このとき, ラグランジュ乗数 λ' と λ の間の差分を $\Delta\lambda$ と定義する. すなわち, $\lambda = \lambda' + \Delta\lambda$ である. 更新されたラグランジュ乗数 λ に対する相対コスト $c_j(\lambda)$ は次のようになる:

$$\begin{aligned} c_j(\lambda) = c_j(\lambda' + \Delta\lambda) &= 1 - \sum_{i \in V} a_{ij}(\lambda'_i + \Delta\lambda_i) \\ &= c_j(\lambda') - \sum_{i \in V} a_{ij} \Delta\lambda_i. \end{aligned}$$

もし $c_j^{\mathrm{UB}}(\lambda') \leq c_j(\lambda') \leq c_j^{\mathrm{LB}}(\lambda')$ が満たされるならば, 式 (4.6) と (4.7) より, 次の不等

式が成り立つ:

$$c_j(\lambda) = c_j(\lambda' + \Delta\lambda) \leq c_j^{\mathrm{UB}}(\lambda') - \sum_{\substack{i \in V \\ \Delta\lambda_i < 0}} a_i^{\max} \Delta\lambda_i - \sum_{\substack{i \in V \\ \Delta\lambda_i \geq 0}} a_i^{\min} \Delta\lambda_i,$$

$$c_j(\lambda) = c_j(\lambda' + \Delta\lambda) \geq c_j^{\mathrm{LB}}(\lambda') - \sum_{\substack{i \in V \\ \Delta\lambda_i \geq 0}} a_i^{\max} \Delta\lambda_i - \sum_{\substack{i \in V \\ \Delta\lambda_i < 0}} a_i^{\min} \Delta\lambda_i.$$

これらの結果から, 次のように更新することで, 現在のラグランジュ乗数 λ に対する相対コストの上界 $c_j^{\mathrm{UB}}(\lambda)$ と下界 $c_j^{\mathrm{LB}}(\lambda)$ を得ることができる:

$$c_j^{\mathrm{UB}}(\lambda) := c_j^{\mathrm{UB}}(\lambda') - \sum_{\substack{i \in V \\ \Delta\lambda_i < 0}} a_i^{\max} \Delta\lambda_i - \sum_{\substack{i \in V \\ \Delta\lambda_i \geq 0}} a_i^{\min} \Delta\lambda_i, \tag{4.8}$$

$$c_j^{\mathrm{LB}}(\lambda) := c_j^{\mathrm{LB}}(\lambda') - \sum_{\substack{i \in V \\ \Delta\lambda_i \geq 0}} a_i^{\max} \Delta\lambda_i - \sum_{\substack{i \in V \\ \Delta\lambda_i < 0}} a_i^{\min} \Delta\lambda_i. \tag{4.9}$$

式 (4.8) と (4.9) より, 相対コスト $c_j(\lambda)$ の真の値の計算を省略したとしても, 最初の反復を除き, SubOpt の各反復で上界 $c_j^{\mathrm{UB}}(\lambda)$ と下界 $c_j^{\mathrm{LB}}(\lambda)$ を得ることができる. ただし, 相対コスト $c_j(\lambda)$ の真の値を計算したときは, 全ての木 $j \in T$ に対して $c_j^{\mathrm{UB}}(\lambda) := c_j(\lambda)$ および $c_j^{\mathrm{LB}}(\lambda) := c_j(\lambda)$ とする.

全ての木 $j \in T$ に対する式 (4.8) と (4.9) よる更新の計算量は $O(|V| + |T|)$ であり, 他のステップの計算量に比べて十分小さい. 全ての頂点 $i \in V$ に対して $a_i^{\max}$ と $a_i^{\min}$ を求める計算量は $O(|V||T|)$ ある. しかし, 全ての頂点 $i \in V$ に対する $a_i^{\max}$ と $a_i^{\min}$ は, SubOpt が呼び出される度に初めに 1 回だけ計算すればよい. この計算量は SubOpt の 1 回目の反復において全ての木 $j \in T$ に対して相対コスト $c_j(\lambda)$ の真の値を計算する時間と同じであり, 十分小さいといえる. さらに, 全ての頂点 $i \in V$ に対する $a_i^{\max}$ と $a_i^{\min}$ を得る計算量をヒープを使用することで削減することができる. すなわち, 全ての頂点 $i \in V$ に対して $a_i^{\max}$ と $a_i^{\min}$ を保持するためのヒープを使用し, 新たな木が追加される度に, および不必要と判断された木が削除される度にヒープを更新する. ヒープを用いた計算量は提案アルゴリズムの実行全体で $O(|V||T| \log |T|)$ 時間になり, SubOpt の呼び出し回数に依存しない. 4.6 節の計算実験ではヒープを用いた実装を使用している. 以上より, 全ての木 $j \in T$ に対する上界 $c_j^{\mathrm{UB}}(\lambda)$ と下界 $c_j^{\mathrm{LB}}(\lambda)$ の計算は SubOpt の各反復においてボトルネックにならない. また, 上で示した上界 $c_j^{\mathrm{UB}}(\lambda)$ と下界 $c_j^{\mathrm{LB}}(\lambda)$ は単純なものであるが, SubOpt の計算時間を削減す

るために非常に効果的であることを観測した. 実際に, 予備実験より, 上記の手法を取り入れた実装の方が取り入れてない法に比べて, SubOpt の 1 回の呼び出しにかかる計算時間が平均的に 2 倍から 4 倍速くなった.

4.4 木パッキングアルゴリズム

前節では, ラグランジュ緩和に基づく 1 段階目の木を生成するアルゴリズム LR-GenInTrees を提案した. 本節では, 2 段階目のアルゴリズム (すなわち, 各木 $j \in T$ のパッキング回数 x_j を決定するアルゴリズム) について議論する. ラグランジュ緩和に基づくアルゴリズムでも 2.4.5 節で示した線形緩和に基づくアルゴリズムと同様に PackInTrees アルゴリズムを使用する. 線形緩和に基づくアルゴリズムでは, 2.4.1 節で示したように式 (2.7) で定義した S' 内の解を PackInTrees アルゴリズムの初期解として利用した. また, 2.6 節の計算実験により, S' 内の解は PackInTrees アルゴリズムの初期解として非常に有益であることを示した. しかし, LRGenInTrees アルゴリズムでは GenInTrees アルゴリズムと異なり, 線形緩和問題 $LP(T)$ の最適解の集合 S を出力しておらず, S' を初期解集合として利用できない. そこで, 本章のラグランジュ緩和に基づくアルゴリズムでも S' と同様の初期解を得るために, LRGenInTrees アルゴリズムが出力した木集合 T のいくつかの部分集合 $T_k \subseteq T$ に対して, 線形緩和問題 $LP(T_k)$ を解いて最適解 x^* を得る.

具体的には, 次のような方法で初期解を得る. T_0 を初期木集合とし, $k = 1, \ldots, |T|$ に対して LRGenInTrees アルゴリズムの k 番目の反復後に得られた木集合を T_k とする. ここで, T は LRGenInTrees アルゴリズムが出力した木集合である. すなわち, $T = T_{|T|}$ である. 提案アルゴリズムは η 個の木集合 $T_{|T|-\eta}, \ldots, T_{|T|}$ に対する線形緩和問題 $LP(T_{|T|-\eta}), \ldots, LP(T_{|T|})$ を解き, η 個の最適解を得る. ここで, η は計算する最適解の数を定めるパラメータである. 得られた η 個の最適解を $\hat{S}$ とする. 式 (2.7) と同様に, この $\hat{S}$ に含まれる線形緩和問題の最適解の小数部を切り捨てて得られた実行可能解の集合を $\hat{S}'$ と定義する. すなわち, $\hat{S}' = \{(\lfloor x_j^* \rfloor \mid j \in T) \mid (x_j^* \mid j \in T) \in \hat{S}\}$ である. ラグランジュ緩和に基づくアルゴリズムでは $\hat{S}'$ 内の実行可能解を PackInTrees アルゴリズムの初期解として使用する.

上記の方法により, ラグランジュ緩和に基づくアルゴリズムでも良い実行可能解

を得ることが期待できる. しかし, η の値を大きくしすぎると, 結果的に線形緩和問題を解くのに費やす計算時間が長くなってしまい, ラグランジュ緩和に基づいて木を生成した意義が薄れてしまう. そこで, 本研究では十分小さな値である $\eta := 10$ とした. この程度の値でも精度の良い解を生成できることが確認できている (4.6 節を参照).

4.5 アルゴリズムの全体像

これまでの節で, 1 段階目のアルゴリズムとして LRGenInTrees アルゴリズムを提案し, 2 段階目のアルゴリズムとして 2.4 節で提案した PackInTrees アルゴリズムを使用することを示した. 本節では提案アルゴリズムの全体像を簡単にまとめる. 提案アルゴリズムは最初に LRGenInTrees アルゴリズムを呼び出し, 木の候補 T を生成する. LRGenInTrees アルゴリズムの内部では良いラグランジュ乗数 λ を得るために SubOpt を呼び出す. 次に, η 個の部分木集合 $T_k \subseteq T$ に対する線形緩和問題 $LP(T_k)$ を解いて, その最適解 x^* の集合 $\hat{S}$ を得る. その後, $\hat{S}$ 内の各解の小数部を切り捨てて得られた実行可能解の集合 $\hat{S}'$ を生成し, η 個の実行可能解に対して PackInTrees アルゴリズムを適用して解の改善を行う. 最後に, PackInTrees アルゴリズムが出力した η 個の実行可能解の中で, 最良の実行可能解を提案アルゴリズムの解として出力する.

4.6 計算実験

4.6.1 計算環境

本章の計算実験では 2.6 節と同様に, 2.6.1 節で示した 2 種類の問題例に対して実験を行った. すなわち, センサーネットワークに関する研究 [17, 26] で使用された, センサーの位置情報から生成された問題例と, ランダムに生成された問題例を使用した. 具体的には, センサーの位置情報から生成された問題例 hcb100, sfis100-1, sfis100-2, sfis100-3 と, 2.6 節で使用したパラメータ $n = 100$ と $\delta = 5\%, 50\%$ と $b = 10000, 100000$ を持つ 12 個の問題例に加えて, 新たに頂点数が 200 である ($n = 200$ の) 問題例を 12

個生成し, 合計24個のランダムに生成された問題例を使用した. なお, 問題例の詳細な生成の方法は2.6節と同じである.

計算環境も2.6.1節で示したものと同様である. 提案アルゴリズムをC++言語で実装し, Dell PowerEdge T300 (CPU: Xeon X3363 2.83GHz, キャッシュ: 6MB, メモリ: 24GB) で計算実験を行った. 線形計画ソルバーはフリーで公開されているGLPK4.43を使用し, 解法にはシンプレックス法を用いた.

4.6.2 計算結果

図4.1と4.2は4.3.6節で述べたSubOptで実際に使用する木の削減による高速化手法を組み込んでいないときの, 問題例sfis100-1とrnd200-50-100000-hに対するSubOptで使用された木の推移を表している. 図4.1と4.2の横軸はLRGenInTreesアルゴリズムによって生成された木を表し, 左から右に生成された順に木が並んでいる (すなわち, LRGenInTreesアルゴリズムのk番目に生成された木を$j_k \in T$とすると, 左から右にj_1から$j_{|T|}$までを並べたものを意味する). 図4.1と4.2の縦軸はLRGenInTreesアルゴリズムの反復を表し, 上から下にアルゴリズムの反復数が増える. 図4.1と4.2では, LRGenInTreesアルゴリズムのある反復において呼び出されたSubOptの中で木j_kが1度でも使用されたとき (1度でも$x_{j_k}(\lambda) > 0$を満たすとき) 黒い点を描いた. 一方, 空白部分はSubOptの中でその木が1度も使用されなかった (SubOptの1回の呼び出しの全ての反復で$x_j(\lambda) = 0$を満たす) ことを表す. ただし, 図4.1と4.2の右上部分の空白領域は, LRGenInTreesアルゴリズムのその反復においてまだ木が生成されてないことを意味する.

図4.1と4.2より, 4.3.6節で述べたように初期木および列生成法の早い段階で生成された木のほとんどは, 後半の劣勾配法では使用されないことが観測できる. とくに初期木 (図4.1と4.2の左側部分) に対してはその傾向が顕著である. また, 一部の初期木を除いて生成された木はしばらくの間SubOptの中で使用され続けることが分かり, 1度だけSubOptの中で使用されなかったからといって, すぐにそのような木を削除するのは危険であることが観測できる. そのため, 4.3.6節では木を削除するまでに十分な期間をおいて, 必要な木まで削除しない方法を提案した.

図4.3と4.4はLRGenInTreesとGenInTreesアルゴリズムに問題例sfis100-1とrnd200-50-100000-hを入力したときの挙動を表している. “Lag.”がラグランジュ

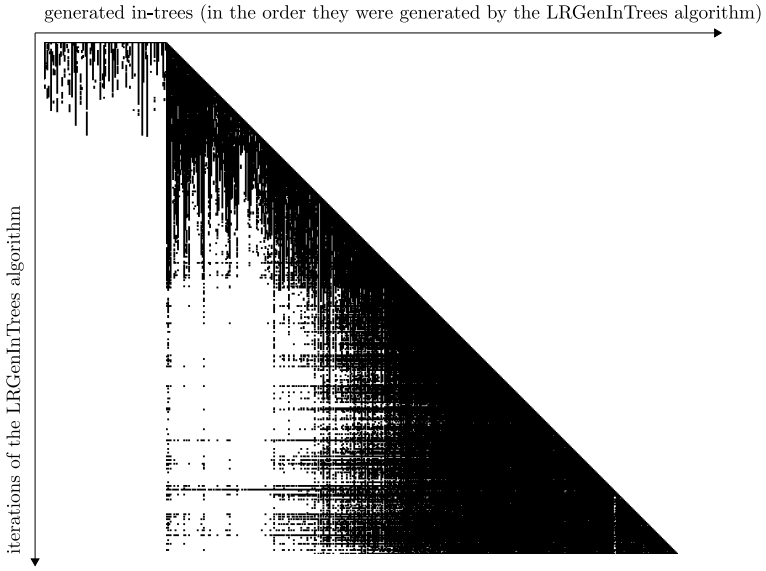


図 4.1: 問題例 sfis100-1 に対する劣勾配法 (木の削減をしていない SubOpt) で使用された木の推移

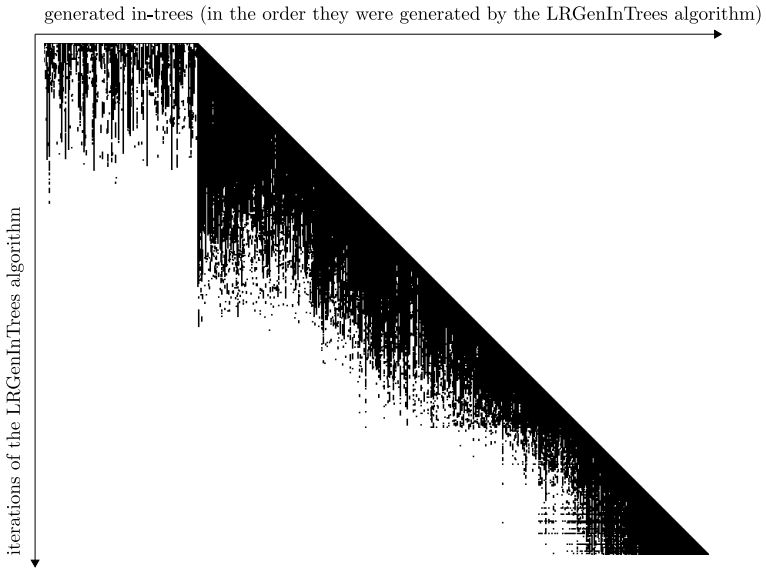


図 4.2: 問題例 rnd200-50-100000-h に対する劣勾配法 (木の削減をしていない SubOpt) で使用された木の推移

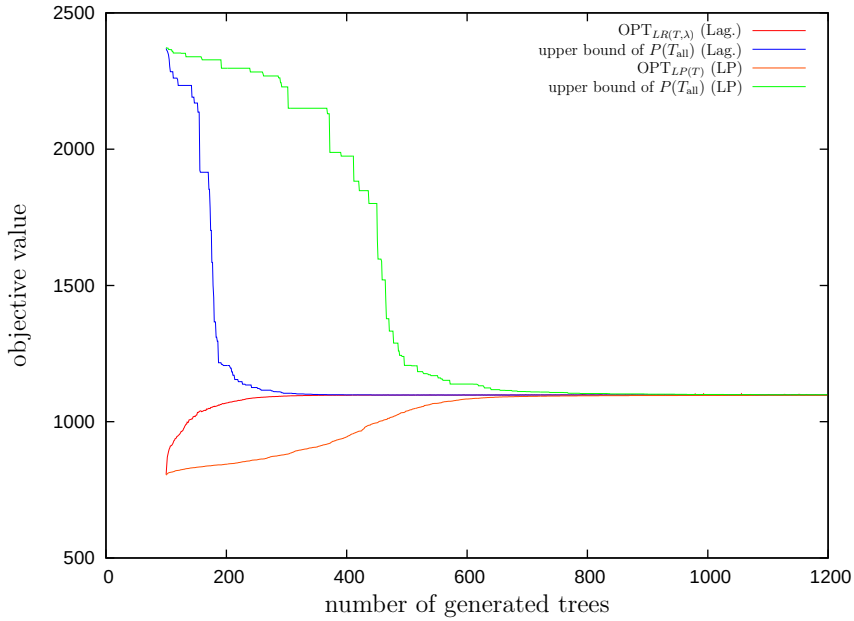


図 4.3: LRGenInTrees と GenInTrees アルゴリズムに問題例 sfis100-1 を入力したときの挙動

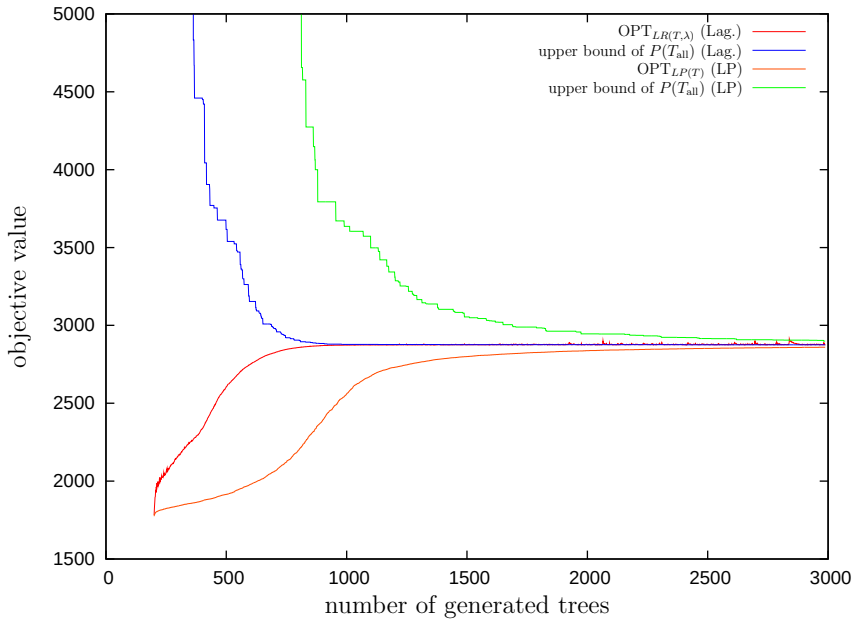


図 4.4: LRGenInTrees と GenInTrees アルゴリズムに問題例 rnd200-50-100000-h を入力したときの挙動

緩和に基づく LRGenInTrees アルゴリズム, “LP”が線形緩和に基づく GenInTrees アルゴリズムを意味する. 図の横軸は生成された木の数, 縦軸は目的関数値である. すなわち, 左から右へアルゴリズムの反復が進む. 図 4.3 と 4.4 で示した LRGenInTrees アルゴリズムは, 実際には 3000 本の木を生成する前に 4.3.4 節で示した停止条件を満たしてしまうが, 図の描画のために木の生成を続けている. 図 4.3 と 4.4 は各反復で木集合 T に新しい木が追加されたときの, ラグランジュ緩和問題の最適値 $\mathrm{OPT}_{LR(T,\lambda)}$, 線形緩和問題の最適値 $\mathrm{OPT}_{LP(T)}$ および元問題 $P(T_{\mathrm{all}})$ の上界 UB^* の改善の様子を示している. 図 2.1 と 2.2 と同様に, LRGenInTrees アルゴリズムにおいても, 反復が進むごとに最適値 $\mathrm{OPT}_{LR(T,\lambda)}$ と上界 UB^* の間の差は小さくなり, 最終的に非常に接近した値となる. しかし, 改善の比率は小さくなっていく. また図 4.3 と 4.4 より, LRGenInTrees アルゴリズムの方が GenInTrees アルゴリズムよりも速く収束していることが分かる. すなわち, 元問題 $P(T_{\mathrm{all}})$ に対する良い上界を得るために必要な木を, LRGenInTrees アルゴリズムは GenInTrees アルゴリズムよりも速く生成できる. 良い上界を与える木集合は元問題 $P(T_{\mathrm{all}})$ に対する良い実行可能解を得るのに有益である傾向にあった. 他の問題例に対してもこのような傾向が観測される.

表 4.1 は本章で提案したラグランジュ緩和に基づくアルゴリズムの計算結果である. また, 第 2 章で提案した線形緩和に基づくアルゴリズムと, 既存研究である Sasaki ら [26] のアルゴリズム, および 2.6.2 節で示した彼らのアルゴリズムを新たに実装したものによる計算結果も併せて示す. 最初の 3 列は, 問題例の名前, 根を除いた頂点数 $|V^-|$, 辺数 $|E|$ を表す. 列 “$\mathrm{UB}_{\mathrm{b.k.}}$” は第 2 章の線形緩和に基づくアルゴリズムを, 長時間動作させて得られた元問題 $P(T_{\mathrm{all}})$ に対する既存の最良の上界を表す. 続く残りの列において, “Lagrangian Relaxation Based” は本章で提案したラグランジュ緩和に基づくアルゴリズム, “LP Relaxation Based” は第 2 章で提案した線形緩和に基づくアルゴリズム, “Sasaki et al. (2007)” は既存研究である Sasaki ら [26] のアルゴリズム, “SFIS” は彼らのアルゴリズムを新たに実装したものを意味する. 列 “$|T|$” は LRGenInTrees アルゴリズムによって生成された木の数, 列 “UB” は LRGenInTrees アルゴリズムによって得られた元問題 $P(T_{\mathrm{all}})$ の最適値 $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ に対する最良の上界, 列 “Obj.” は提案アルゴリズムが出力した目的関数値, 列 “Gap (%)” は $\mathrm{UB}_{\mathrm{b.k.}}$ と Obj. の間のギャップ (すなわち, $((\mathrm{UB}_{\mathrm{b.k.}} - \mathrm{Obj.})/\mathrm{UB}_{\mathrm{b.k.}}) \times 100\%$),

列 “Time (s)” は計算時間 (秒) を表す．列 “Sasaki et al. (2007)” における “–” は結果が得られていないことを意味する．なお, 表 4.1 の実験では, LRGenInTrees アルゴリズムが GenInTrees アルゴリズムよりどれだけ速く良い木集合を生成できるかを示すために, GenInTrees アルゴリズムは本来の停止条件を満たす前に, LRGenInTrees アルゴリズムが生成した木の数 (表中の列 “$|T|$”) と同じ数の木を生成した時点で停止するようにしている．また, 新たな実装 SFIS に対しては計算時間が同程度となるように $|V^-| = 100$ の問題例に対して 10 秒の時間制限, $|V^-| = 200$ の問題例に対して 100 秒の時間制限を与えている．

表 4.1 より, 本章で提案したアルゴリズムは第 2 章で提案したアルゴリズム, Sasaki ら [26] のアルゴリズム, および SFIS よりも良い結果が得られたことを示している．生成した木の数が同じであり, (頂点数が $|V^-| = 100$ であるいくつかの問題例を除いて) 計算時間が短いにも関わらず, 本章で提案したアルゴリズムは第 2 章で提案したアルゴリズムよりも良い目的関数値を達成できたことが観測できる．頂点数が $|V^-| = 200$ である問題例に対して, 本章で提案したアルゴリズムの計算時間は平均約 60 秒であり, 生成された木の数は平均約 $7|V^-|$ 本である．また, $\mathrm{UB}_{\mathrm{b.k.}}$ と目的関数値のギャップは非常に小さく, とくに容量が $b = 100000$ である (rnd200-50-100000 を除く) 問題例に対して, ギャップは 1%よりも小さい．さらに, 本章で提案したアルゴリズムは 3 つの問題例に対して厳密な最適解を発見した．

表 4.2 はラグランジュ緩和に基づくアルゴリズムの出力した目的関数値に達成するのに必要な, 線形緩和に基づくアルゴリズムの木の生成数と計算時間を示している．表 4.2 の各列の意味は, 列 “Time (s)” を除いて表 4.1 と同じである．最初の 5 列は表 4.1 と同じものである．列 “Time (s)” は 1 段階目の木を生成する LRGenInTrees と GenInTrees アルゴリズムの計算時間を表す．すなわち, 2 段階目の実行可能解を生成する計算時間は含まれていない．残りの 3 列は, 本章で提案したアルゴリズムの出力した目的関数値に達成したときの, 第 2 章で提案したアルゴリズムの計算結果である．ただし, 列 “Obj.” の記号 “$*$”は, 第 2 章で提案したアルゴリズムが停止条件を満たすまでに, 本章で提案したアルゴリズムの出力した目的関数値に達成できなかったことを表す．“$*$”の付いた結果は, 第 2 章で提案したアルゴリズムが本来の停止条件で終了したときのものである．表 4.2 より, 本章で提案したアルゴリズムと同等の結果を第 2 章で提案したアルゴリズムが得るためには, 数多くの木を生成し

表 4.1: 提案および既存アルゴリズムの計算結果

Instance name	$\|V^-\|$	$\|E\|$	$UB_{b.k.}$	Lagrangian Relaxation Based					LP Relaxation Based			Sasaki et al. (2007)	SFIS
				$\|T\|$	UB	Obj.	Gap (%)	Time (s)	Obj.	Gap (%)	Time (s)		
hcb100	100	10100	1124	684	1125	1119	0.44	6.2	1092	2.85	6.0	–	930
sfis100-1	100	10100	1097	695	1098	1089	0.73	8.4	1081	1.46	6.1	961	1032
sfis100-2	100	10100	1097	556	1098	1090	0.64	5.8	1059	3.46	3.9	969	1032
sfis100-3	100	10100	1101	696	1102	1095	0.54	8.8	1089	1.09	6.6	1022	1021
rnd100-5-10000-h	100	473	225	627	225	216	4.00	4.1	181	19.56	5.7	–	159
rnd100-5-10000-t	100	473	217	788	229	217	0.00	7.1	191	11.98	7.5	–	209
rnd100-5-10000	100	473	128	608	128	123	3.91	3.8	108	15.63	5.4	–	99
rnd100-5-100000-h	100	473	2251	605	2252	2243	0.36	5.3	1874	16.75	5.2	–	1611
rnd100-5-100000-t	100	473	2173	869	2302	2173	0.00	12.0	2046	5.84	9.2	–	2108
rnd100-5-100000	100	473	1283	691	1283	1276	0.55	5.4	1160	9.59	7.1	–	1003
rnd100-50-10000-h	100	4938	272	931	272	263	3.31	10.4	261	4.04	10.9	–	211
rnd100-50-10000-t	100	4938	270	656	270	257	4.81	7.1	186	31.11	6.3	–	238
rnd100-50-10000	100	4938	149	542	149	143	4.03	4.2	132	11.41	3.8	–	119
rnd100-50-100000-h	100	4938	2726	784	2726	2717	0.33	9.9	2673	1.94	7.8	–	2292
rnd100-50-100000-t	100	4938	2701	628	2701	2688	0.48	9.2	1945	27.99	6.0	–	2413
rnd100-50-100000	100	4938	1498	597	1499	1490	0.53	5.1	1430	4.54	4.6	–	1295
rnd200-5-10000-h	200	1970	260	1065	260	247	5.00	20.4	185	28.85	70.9	–	172
rnd200-5-10000-t	200	1970	250	1323	261	249	0.40	39.1	183	26.80	104.0	–	229
rnd200-5-10000	200	1970	141	898	141	131	7.09	15.4	104	26.24	55.6	–	106
rnd200-5-100000-h	200	1970	2602	1451	2603	2583	0.73	56.7	2152	17.29	149.8	–	1755
rnd200-5-100000-t	200	1970	2500	1291	2622	2500	0.00	52.8	1943	22.28	101.0	–	2302
rnd200-5-100000	200	1970	1411	1005	1412	1401	0.71	22.7	1173	16.87	70.3	–	1091
rnd200-50-10000-h	200	20030	287	1652	287	273	4.88	73.3	265	7.67	153.6	–	200
rnd200-50-10000-t	200	20030	286	1329	286	273	4.55	58.9	179	37.41	109.0	–	250
rnd200-50-10000	200	20030	156	987	157	146	6.41	29.0	107	31.41	51.8	–	111
rnd200-50-100000-h	200	20030	2874	1525	2876	2857	0.59	76.5	2779	3.31	123.9	–	2374
rnd200-50-100000-t	200	20030	2867	1270	2868	2855	0.42	78.1	1920	33.03	100.9	–	2542
rnd200-50-100000	200	20030	1569	929	1570	1552	1.08	28.8	1299	17.21	48.2	–	1332

て長い計算時間をかけなければならないことが分かる. 頂点数が $|V^-| = 200$ である問題例に対してはこの傾向がとくに顕著である.

なお, 本章および第2章で提案したアルゴリズムは1段階目の木を生成する処理と, 2段階目の各木のパッキング回数を決定する処理を分離していたので, 目的関数値による列生成法の停止条件は本来導入できない. そこで, 表4.2の結果を得るために第2章で提案したアルゴリズムを修正した. GenInTrees アルゴリズムの各反復において新たな木が木集合 T に追加される度に, 実行可能解を生成する (すなわち, 線形緩和問題 $LP(T)$ の最適解 $(x_j^* \mid j \in T)$ に対する小数部を切り捨てた実行可能可能 $(\lfloor x_j^* \rfloor \mid j \in T)$ に PackInTrees アルゴリズムを適用して改善した解を生成する). この修正のため, 表4.2の第2章で提案したアルゴリズムの "$*$" の付いた結果は, 表2.2の結果と少し異なる. また, この修正により PackInTrees アルゴリズムの呼び出し回数が大幅に増えてしまうため, 全体の計算時間が元の実装よりも長くなってしまう. そのため, 表4.2では2段階目の PackInTrees アルゴリズムに費やした時間を除いた計算時間を報告している.

4.7 結論

本章では, 頂点容量制約付き有向全域木パッキング問題に対するラグランジュ緩和に基づく発見的解法を提案した. 提案アルゴリズムは2段階で構成されており, まず候補となる木の集合 T を生成し, 次に生成した各木 $t \in T$ に対してパッキング回数を決定する. 1段階目では, ラグランジュ緩和問題 $LR(T, \lambda)$ に対する良いラグランジュ乗数 λ を得るために劣勾配法を適用し, 得られたラグランジュ乗数 λ に基づいて列生成法を適用することで, 繰り返し新たな木を生成して T に追加する方法を提案した. また, 劣勾配法に対して, 使用する木を削減したり, 各反復における実際の計算量を削減する高速化手法を提案した. 2段階目は, 第2章で提案したアルゴリズムによって得られた実行可能解の質が良いことから, 同様の方法を用いることにした. 本章で提案したアルゴリズムは, 第2章で提案したアルゴリズムに比べて, 同じ木の数を生成するのに要する計算時間が短いにも関わらず, より良い解を出力できることを示した. また, 一部の問題例に対しては厳密な最適解を得ることができた. 本章で提案したアルゴリズムは頂点容量制約付き有向全域木パッキング問題に対し

表 4.2: ラグランジュ緩和に基づくアルゴリズムの出力した目的関数値に達成するのに必要な, 線形緩和に基づくアルゴリズムの木の生成数と計算時間

Instance name	$\|V^-\|$	$\|E\|$	Lagrangian Relaxation Based			LP Relaxation Based		
			$\|T\|$	Obj.	Time (s)†	$\|T\|$	Obj.	Time (s)†
hcb100	100	10100	684	1119	5.9	1501	1119	27.4
sfis100-1	100	10100	695	1089	8.0	944	1089	10.8
sfis100-2	100	10100	556	1090	5.5	952	1090	11.1
sfis100-3	100	10100	696	1095	8.3	1036	1095	13.8
rnd100-5-10000-h	100	473	627	216	3.7	1492	216	23.8
rnd100-5-10000-t	100	473	788	217	7.0	1227	206*	14.7
rnd100-5-10000	100	473	608	123	3.4	1520	123	25.2
rnd100-5-100000-h	100	473	605	2243	4.9	2007	2243	40.3
rnd100-5-100000-t	100	473	869	2173	11.9	1227	2162*	14.7
rnd100-5-100000	100	473	691	1276	4.9	1963	1276	39.1
rnd100-50-10000-h	100	4938	931	263	9.7	1120	263	14.7
rnd100-50-10000-t	100	4938	656	257	6.6	1688	256*	23.7
rnd100-50-10000	100	4938	542	143	3.9	1096	143	14.2
rnd100-50-100000-h	100	4938	784	2717	9.5	1916	2717	40.8
rnd100-50-100000-t	100	4938	628	2688	8.7	1705	2688	24.4
rnd100-50-100000	100	4938	597	1490	4.7	1645	1490	30.6
rnd200-5-10000-h	200	1970	1065	247	17.5	4408	247	1474.9
rnd200-5-10000-t	200	1970	1323	249	38.3	5422	237*	1584.1
rnd200-5-10000	200	1970	898	131	13.2	3591	131	984.2
rnd200-5-100000-h	200	1970	1451	2583	52.3	6799	2583	3547.5
rnd200-5-100000-t	200	1970	1291	2500	52.6	5422	2485*	1583.3
rnd200-5-100000	200	1970	1005	1401	20.0	7224	1401	4105.4
rnd200-50-10000-h	200	20030	1652	273	68.3	2563	273	441.1
rnd200-50-10000-t	200	20030	1329	273	54.6	6062	273	1912.2
rnd200-50-10000	200	20030	987	146	26.7	2522	146	431.0
rnd200-50-100000-h	200	20030	1525	2857	71.9	4448	2857	1563.1
rnd200-50-100000-t	200	20030	1270	2855	73.7	7438	2855	2752.1
rnd200-50-100000	200	20030	929	1552	26.8	3699	1552	1073.5

† 表中の列 “Time (s)” は 1 段階目の木を生成する LRGenInTrees と GenInTrees アルゴリズムの計算時間を表す. すなわち, 2 段階目の実行可能解を生成する計算時間は含まれていない.

て有効であり, 第2章で提案したアルゴリズムよりもさらに実用的であると結論づけられる.

第5章　結論

本研究では, 頂点容量制約付き有向全域木パッキング問題を扱った. 頂点容量制約付き有向全域木パッキング問題はセンサーネットワークなどの実用的な応用を持つが, 強NP困難であることが示されており, 厳密な最適解を求めることは困難である. そこで, 本研究では頂点容量制約付き有向全域木パッキング問題に対する2つの発見的解法を提案した.

1つ目は第2章で提案した線形緩和に基づくアルゴリズムである. このアルゴリズムは2段階で構成されており, まず候補となる木の集合Tを生成し, 次に生成した各木$t \in T$に対してパッキング回数を決定する. 1段階目では, 線形緩和問題$LP(T)$に列生成法を適用し, 繰り返し新たな木を生成してTに追加する方法を提案した. その中で, これを実現するために解く必要のある価格付け問題が多項式時間で解けることを示した. 2段階目は, 第1段階で生成した線形緩和問題$LP(T)$の最適解xを修正して実行可能解を作り, 貪欲アルゴリズムによって解を改善する. この貪欲アルゴリズムに対して効率的な実装方法を提案し, データ構造を工夫することにより高速化した. 計算実験により, 第2章で提案したアルゴリズムは既存アルゴリズムよりも良い解を出力し, 上界と目的関数値のギャップが非常に小さいことが観測された. すなわち, 第2章で提案したアルゴリズムは頂点容量制約付き有向全域木パッキング問題に対して実用的な時間で精度の高い解を出力できることを確認した.

2つ目は第4章で提案したラグランジュ緩和に基づくアルゴリズムである. 第2章で提案したアルゴリズムと基本的な流れは同じであり, 1段階目に木の集合Tを生成し, 2段階目に各木$t \in T$のパッキング回数を決定する. 1段階目では, ラグランジュ緩和問題$LR(T, \lambda)$に対する良いラグランジュ乗数λを得るために劣勾配法を適用し, 得られたラグランジュ乗数λに基づいて列生成法を適用することで, 繰り返し新たな木を生成してTに追加する方法を提案した. また, 劣勾配法に対して, 使用する木を削減したり, 各反復における実際の計算量を削減する高速化手法を提案した.

2段階目は, 第2章で提案したアルゴリズムによって得られた実行可能解の質が良いことから, 同様の方法を用いた. 計算実験により, 第4章で提案したアルゴリズムは第2章で提案したアルゴリズムおよび既存アルゴリズムに比べて良い解を出力した. とくに, 第2章で提案したアルゴリズムに対しては, 同じ木の数を生成するのに要する計算時間が短いにも関わらず, より良い解を出力できることを示した. すなわち, 第4章で提案したアルゴリズムは第2章で提案したアルゴリズムよりも実用的に優れていることを確認した.

また, 第3章では頂点容量制約付き有向全域木パッキング問題に対する線形緩和に基づくアルゴリズムの近似精度について述べた. 具体的には, 一般化された線形緩和に基づくアルゴリズムは, 最適値 $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ が頂点数 $|V|$ の α 倍 $(\alpha > 1)$ である問題例に対して $(1 - 1/\alpha)$ 近似アルゴリズムであることを示した. この結果より, 第2章で提案したアルゴリズも, 列生成法の停止条件を式 (2.5) のみにしたときには $(1 - 1/\alpha)$ 近似アルゴリズムであることが結論できる.

実際の応用においては, 様々な付加的な制約が必要であることが少なくなく, 本研究で与えた2つのアルゴリズムが直接適用できない問題が多数存在する. しかし, 似通った構造を持つ問題であれば, それぞれの問題に特化したアルゴリズムを設計する上での指針や, 内部で呼び出す部品として本研究の成果を活用することができる. また, 本研究で与えた理論的な結果は, 頂点容量制約付き有向全域木パッキング問題に対する興味深い知見を与えてくれる. これらの成果が, センサーネットワークに関する分野, グラフパッキングに関する分野, さらには最適化分野の発展に貢献できれば幸いである.

謝辞

多くの方々の助けがなければ本研究を遂行することは到底不可能でした．この場を借りて感謝の意を述べたいと思います．

本研究に対して適切かつ詳細な数多くの御指導と御指摘を賜りました指導教員である名古屋大学大学院情報科学研究科の柳浦睦憲先生に深く感謝致します．出来の悪い学生で苦労をかけたと思いますが, 先生の多大なる御尽力により本研究に一つの区切りをつけることができました．また, 研究面だけはなく, 私生活面でも御指導いただき, 先生のおかげで有意義な大学院生活を過ごすことができました．重ねて感謝申し上げます．

優れた洞察力と観察力により御助言を頂いた共同研究者である名古屋大学大学院工学研究科の今堀慎治先生に深く感謝致します．平田研究室に所属していないにもかかわらず頻繁にミーティングに参加して頂き, 本研究を円滑に進めることができました．

本研究の動機づけと御助言を頂いた共同研究者である南山大学情報理工学部の佐々木美裕先生に深く感謝致します．私が本研究のテーマに巡り会えたのは佐々木先生のおかげであるといっても過言ではありません．また, 研究発表の練習の際に有意義な御指導を頂いたことにも感謝致します．

私が平田研究室在籍中に長年に渡り御指導してくださった名古屋大学大学院情報科学研究科の平田富夫先生に深く感謝致します．本研究を円滑に進めるための研究室の環境は平田先生の御尽力によるものです．また, 研究者としての様々な貴重な体験をさせて頂いたことに感謝を申し上げます．

私が博士後期課程に 3 年間在籍している間に, 先輩として絶えず私に丁寧な御助言をくださった名古屋大学大学院情報科学研究科の橋本英樹先生に深く感謝致します．私の詰まらない質問にも丁寧に御返答してくださったことに感謝を申し上げます．橋本先生のおかげで楽しく研究生活を送ることができました．

本博士論文を執筆の際に有意義なコメントをしてくださった名古屋大学大学院情報科学研究科の酒井正彦先生に感謝致します.

本研究を英文論文誌に投稿する際に英語の御指導をしてくださった Universidade Federal de Sergipe の櫻庭・セルソ・智先生に感謝致します. 私の英語力が乏しく非常にご迷惑をかけたと思いますが, 夜遅くまでご丁寧に御指導いただき大変助かりました.

また, 研究以外の面でもいろいろな方にお世話になりました. 平田研究室の皆様, 秘書の岩本悦子さんおよび柳浦先生の奥様に感謝致します. 私が不自由なく研究会等に参加できたのは岩本悦子さんのおかげです. しばしば夜遅くまで柳浦先生とミーテイングをしてしまい, 柳浦先生の奥様には大変ご迷惑をおかけして申し訳ありませんでした.

最後に, 長男でありながら 9 年間も大学に通わせてくれた両親に深く感謝します. 決して特別裕福な家庭ではありませんでしたが, 無事にここまでこられたのは両親のおかげであると思います. 本当にありがとうございました.

参考文献

[1] E. Balas and A. Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study. *Mathematical Programming Study*, 12:37–60, 1980.

[2] J. E. Beasley. Lagrangian relaxation. In C. R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 243–303. Orient Longman, 1993.

[3] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi. Fast edge splitting and Edmonds' arborescence construction for unweighted graphs. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 455–464, 2008.

[4] F. C. Bock. An algorithm to construct a minimum directed spanning tree in a directed network. In B. Avi-Itzak, editor, *Developments in Operations Research*, pages 29–44. Gordon and Breach, New York, 1971.

[5] G. Calinescu, S. Kapoor, A. Olshevsky, and A. Zelikovsky. Network lifetime and power assignment in ad-hoc wireless networks. In *Proceedings of the 11th European Symposium on Algorithms*, volume 2832 of *Lecture Notes in Computer Science*, pages 114–126. Springer, 2003.

[6] A. Caprara, A. Panconesi, and R. Rizzi. Packing cycles in undirected graphs. *Journal of Algorithms*, 48:239–256, 2003.

[7] Y. Chu and T. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

[8] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

[9] J. Edmonds. Edge-disjoint branchings. In B. Rustin, editor, *Combinatorial Algorithms*, pages 91–96. Academic Press, 1973.

[10] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27:1–18, 1981.

[11] S. Fujishige. A note on disjoint arborescences. *Combinatorica*, 30:247–252, 2010.

[12] H. N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50:259–273, 1995.

[13] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica Archive*, 6:109–122, 1986.

[14] H. N. Gabow and K. S. Manu. Packing algorithms for arborescences (and sspanning trees) in capacitated graphs. *Mathematical Programming*, 82:83–109, 1998.

[15] H. N. Gabow and H. H. Westermann. Forests, frames, and games: Algorithms for matroid sums and applications. *Algorithmica*, 7:465–497, 1992.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[17] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1:660–670, 2002.

[18] M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

[19] S. Imahori, Y. Miyamoto, H. Hashimoto, Y. Kobayashi, M. Sasaki, and M. Yagiura. The complexity of the node capacitated in-tree packing problem. *Networks*, 59:13–21, 2012.

[20] N. Kamiyama, N. Katoh, and A. Takizawa. Arc-disjoint in-trees in directed graphs. *Combinatorica*, 29:197–214, 2009.

[21] M. Krivelevich, Z. Nutov, M. R. Salavatipour, J. Yuster, and R. Yuster. Approximation algorithms and hardness results for cycle packing problems. *ACM Transactions on Algorithms*, 3, 2007.

[22] L. Lovász. On two minimax theorems in graph. *Journal of Combinatorial Theory*, 21:96–103, 1976.

[23] W. Mader. On n-edge-connected digraphs. *Combinatorica*, 1:385–386, 1981.

[24] S. Masuyama and T. Ibaraki. Chain packing in graphs. *Algorithmica*, 6:826–839, 1991.

[25] P. A. Pevzner. Branching packing in weighted graphs. *American Mathematical Society Translations*, 2:185–200, 1994.

[26] M. Sasaki, T. Furuta, F. Ishizaki, and A. Suzuki. Multi-round topology construction in wireless sensor networks. In *Proceedings of the Asia-Pacific Symposium on Queueing Theory and Network Applications*, pages 377–384. 2007.

[27] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.

[28] R. E. Tarjan. A good algorithm for edge-disjoint branching. *Information Processing Letters*, 3:51–53, 1974.

[29] P. Tong and E. Lawler. A faster algorithm for finding edge-disjoint branchings. *Information Processing Letters*, 17:73–76, 1983.

[30] S. Umetani and M. Yagiura. Relaxation heuristics for the set covering problem. *Journal of the Operations Research Society of Japan*, 50:350–375, 2007.

研究実績

学術論文

1. Y. Tanaka, S. Imahori, M. Sasaki and M. Yagiura, An LP-Based Heuristic Algorithm for the Node Capacitated In-Tree Packing Problem, *Computers and Operations Research*, 39 (2012), 637–646.

2. Y. Tanaka, S. Imahori and M. Yagiura, Lagrangian-Based Column Generation for the Node Capacitated In-Tree Packing, *Journal of the Operations Research Society of Japan*, 54-4 (2011), 219–236.

国際発表 (査読あり)

1. Y. Tanaka, M. Sasaki, and M. Yagiura, A Heuristic Algorithm for the Node Capacitated In-Tree Packing Problem, *Proceedings of the International Symposium on Scheduling (ISS 2009)*, (2009), 58–64.

2. Y. Tanaka, S. Imahori and M. Yagiura, A Lagrangian Heuristic Algorithm for the Node Capacitated In-Tree Packing Problem, *Proceedings of the 7th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications (HJ2011)*, (2011), 437–446.

3. T. Sugiyama, Y. Tanaka, H. Hashimoto and M. Yagiura, An Algorithm to Find a Scheduling Table for Embedded Systems, *Proceedings of the International Symposium on Scheduling (ISS 2011)*, (2011), 149–154.

国内発表 (査読あり)

1. 田中勇真, 佐々木美裕, 柳浦睦憲, An LP-Based Heuristic Algorithm for the Node Capacitated In-Tree Packing Problem, 第 8 回情報科学技術フォーラム (FIT 2009), 1 (2009), 23–30.

2. 今堀慎治, 簡于耀, 田中勇真, 柳浦睦憲, Enumerating Bottom-Left Stable Positions for Rectangles with Overlap, 第 9 回情報科学技術フォーラム (FIT 2010), 1 (2010), 25–30.

3. 川島大貴, 田中勇真, 今堀慎治, 柳浦睦憲, 3 次元箱詰め問題に対する構築型解法の効率的実現法, 第 9 回情報科学技術フォーラム (FIT 2010), 1 (2010), 31–38.

4. 川島大貴, 田中勇真, 今堀慎治, 柳浦睦憲, 3 次元パッキング問題に対する best-fit 法の効率的実現法, 第 10 回情報科学技術フォーラム (FIT 2011), 1 (2011), 29–36.