修 士 論 文

ヒッチコック型輸送問題に対する途中棄却を含む段階的解法

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

池田 隆之

2003 年 3 月

修 士 論 文

ヒッチコック型輸送問題に対する途中棄却を含む段階的解法

指導教官　浅野 哲夫 教授

審査委員主査　浅野 哲夫 教授
審査委員　中野 浩嗣 助教授
審査委員　平石 邦彦 助教授

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

110007 池田 隆之

提出年月: 2003 年 2 月

概 要

本研究ではヒッチコック型輸送問題を扱う．ネットワークに対し, できるだけ少ない費用で決められた量のものを流す問題を最小費用流問題という．ヒッチコック型輸送問題は, この最小費用流問題の特別な場合の問題であると言える．すなわち, 入力されるグラフが 2 部グラフであり, 一方から他方への有向辺のみが存在する場合である．ヒッチコック型輸送問題は, その名が示すように, 複数の分散した工場で作られた製品を同じように複数の店舗に移動させるための費用を最小にする配送手順を考えることに似ている．また, このような輸送問題とは一見して関係のないような分野でも用いられていることがある．質問画像に似ている画像をデータベースの中から探し出すことを考える．このとき我々は, 質問画像とデータベース内の画像が”似ている”か”似ていない”かを定量的に判定する必要がある．このための方法のひとつとして,Earth Mover's Distance (EMD) と呼ばれる距離関数を用いた方法がある．この方法では, まず, 比較する画像をカラーシグネチャと呼ばれる表現方法で表すことを行う．カラーシグネチャとは画像を色空間上の重みつき点集合として表現する方法である．それぞれのカラーシグネチャを EMD に入力として与えると, 画像間の相違度に応じた距離を出力として得られる．この EMD を求めるためには, その内部でヒッチコック型輸送問題を解く必要がある．このようにヒッチコック型輸送問題には様々な応用がある．

この論文では, ヒッチコック型輸送問題の解における最少の費用と, 閾値との比較を行う判定問題のアルゴリズムを提案する．上述した EMD の応用では, 画像間の正確な距離を知ることはさほど重要ではない．むしろ, その距離があらかじめ設定された閾値を超えるかどうかを知ることが重要である．上述した EMD では, 設定された閾値以下の距離を持つ画像同士は似ていると判断され, そうでない場合は似ていないと判断される．さらに, まったく似ていない画像との比較を行う場合でも, 判定が困難であるような場合, すなわちその距離が閾値に近い値であるような場合と同様の計算時間が必要になるという問題点がある．そこで本研究では, 最適解を求めるよりも少ない計算時間でヒッチコック型輸送問題の最小の費用の下界を求め, 閾値との比較を段階的に行う手法を提案する．各段階で比較を行い, 最小の費用の下界が閾値よりも大きいことがわかればその段階で棄却する．これによって最少の費用と閾値との間に大きな差がある場合には, より早い段階で”似ていない”と判断され, 棄却される．この結果, 平均的な計算時間は短くなると予想される．

一般のヒッチコック型輸送問題では下界を求めることが困難であることから, 本論文ではユークリッド空間上の点集合を考え,2 点間の単位流量当りの費用がその距離で与えられると仮定する．実際の輸送問題や, 上述した EMD ではこの仮定は成り立つ．よって, 今後はこの仮定の下で議論を進める．提案手法ではまず, 入力されたグラフを縮約し, 問題の規模を小さくする．具体的には, グラフに存在する任意の 2 点を取り出し, その重心を座標とする点でこれらを置き換える．すべての点についてこの操作を行う．これを一段階での縮約とし, 点の数がただひとつとなるまで操作を繰り返す．点の数が少ないほうから各段階

についてヒッチコック型輸送問題を解く. 得られた最小の費用と閾値の比較を行う. 閾値よりも最小の費用が大きい場合にはその時点で棄却を行う. そうでなければ次の段階についてヒッチコック型輸送問題を解き, 最小の費用を得る. 縮約されたグラフにおけるヒッチコック型輸送問題の最小の費用が, 縮約を行う前のグラフにおける最小費用の下界を与えることを示す. これにより, 提案手法の正しいことを述べる. 最小の費用と閾値の差が小さい場合を考える. これは, 縮約されたグラフでは正しく判定を行えないことを意味している. すべての段階を順次計算し, 最初に与えられた問題を解くことになっても計算量が増加しないことを示す.

提案手法のアルゴリズムの実装を行った. 記述はC++言語を用いて行った. また,LEDAをあわせて利用した. 計算機で実際に動作を行った. 乱数を用いて発生させた点を用いてシュミレーションを行った. 点の数を, １０, ２０, ４０, ８０, １６０, ３２０とし, それぞれについて 100 回の比較の平均時間を計測した. その結果, 棄却率が高い場合には計算時間はとても早くなることがわかった.

今回の研究では縮約を行う 2 点は任意のものとしていたが, これに条件を加えることで性能の向上をめざす. 画像データベースへの応用を行い, これまでの方法との比較を行いたいと考えている.

目 次

第1章　まえがき

ここでは, 本論文の主要なテーマであるヒッチコック型輸送問題について述べる. まず, 簡単な例とともにヒッチコック型輸送問題がどのような問題であるかについて述べる. ここではデータを記憶装置に記録する場合の最適化問題を例にとる. 次にこの問題に関係する分野での先行研究について述べる. ヒッチコック型輸送問題を包含する最小費用流問題のための効率的なアルゴリズムや, ヒッチコック型輸送問題に対するアルゴリズムを紹介する. その後, 本研究での提案手法について概要と目的を述べる. 提案手法は特に, ヒッチコック型輸送問題での判定問題に着目して開発されたことや, 多くの場合に計算時間を短縮できることについて触れる. 最後に本論文の構成について述べる.

1.1　ヒッチコック型輸送問題について

記憶装置にデータを記録することを考えよう. k 個の記憶装置 $D_1, D_2, ..., D_k$ と n 個のデータ $z_1, z_2, ...z_n$ からなるシステムを構築する. 記憶装置 D_i のサイズは λ_i であり, データ z_j のサイズは w_j であるとする. データはそれぞれプロセッサーから呼び出され, 記憶装置に配置される. データ z_j を一単位呼び出し, 記憶装置に配置するのに必要な時間は $\alpha_{i,j}$ であることがわかっている. この様子を図 1.1 に示す. 全てのデータを記憶装置に記録する. ただし, この時の通信費用を最小にしたい. このような問題はヒッチコック型輸送問題と呼ばれる, 線形計画問題の一つである. また今回の例を定式化すれば次のようになる.

目的関数

$$Minimize \quad \sum_{i=1}^{k}\sum_{j=1}^{n}\alpha_{i,j}y_{i,j}$$

制約条件

$$\sum_{j=1}^{n} y_{i,j} = \lambda_i \quad (i = 1, 2, ..., k)$$

$$\sum_{i=1}^{k} y_i, j = \omega_j \quad (j = 1, 2, ..., n)$$

$$y_{i,j} \geq 0 \quad (i = 1, 2, ..., k; j = 1, 2, ..., n)$$

これを見ると, 良く知られた線形計画問題の一つとして表されていることがわかる.

さらに, ヒッチコック型輸送問題は最小費用流問題のひとつに分類される. これまでに述べたヒッチコック型輸送問題の位置付けについて図 1.2 に示す. ヒッチコック型輸送問

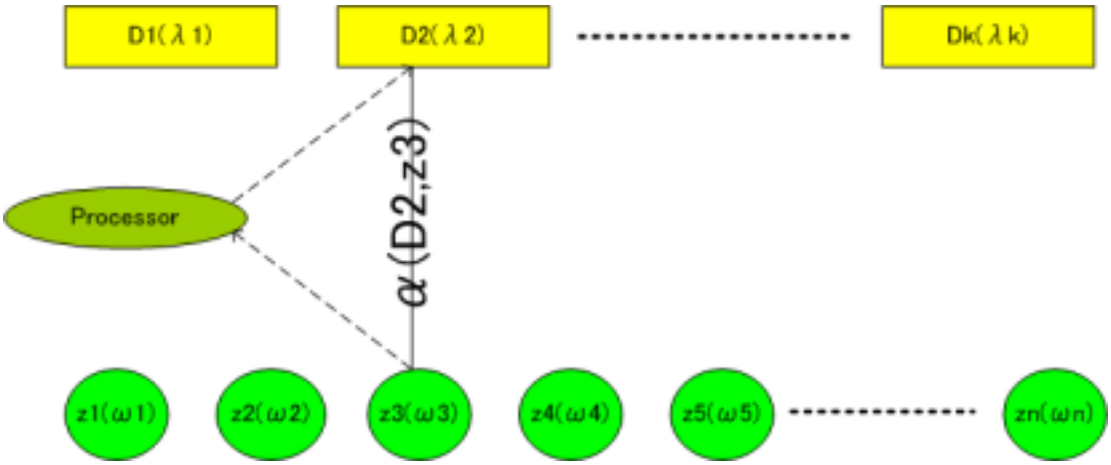

図 1.1: ヒッチコック型輸送問題の例

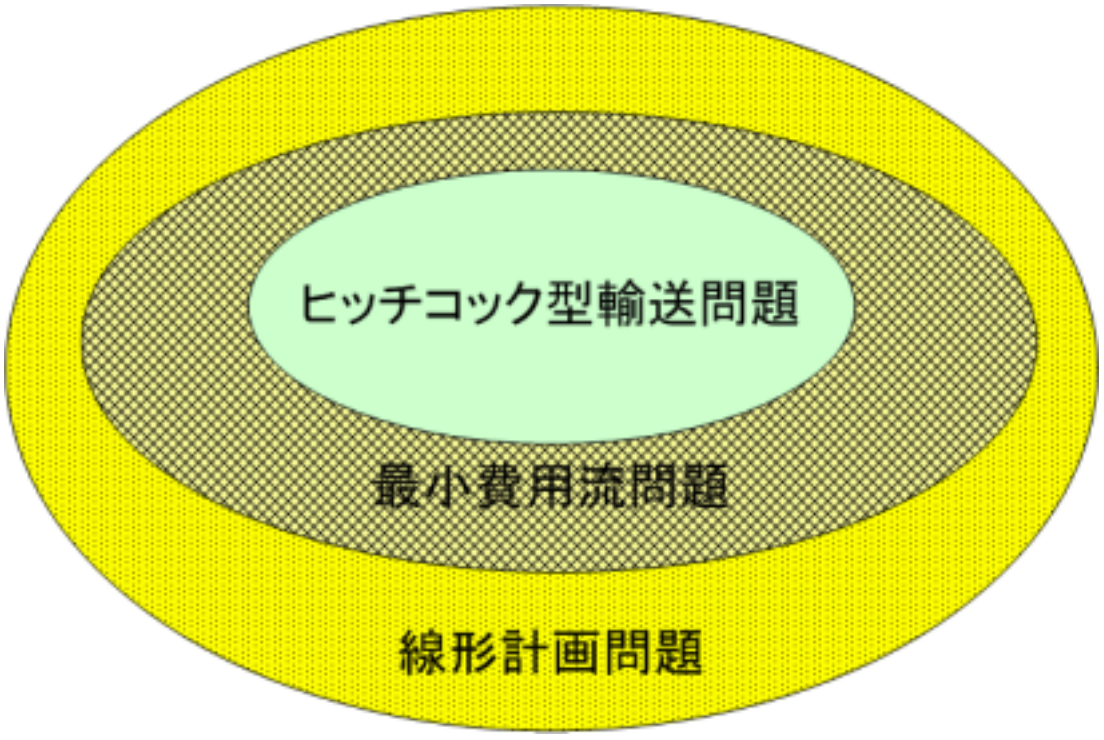

図 1.2: ヒッチコック型輸送問題の位置付け

表 1.1: 最小費用流問題に対する強多項式アルゴリズム

Tardos [1985]	$O(m^4)$
Orlin [1984]	$O\left((n+m')^2 S(n,m)\right)$
Fujishige [1986]	$O\left((n+m')^2 S(n,m)\right)$
Galil and Tardos [1986]	$O(n^2 \log n S(n,m))$
Goldberg and Tarjan [1987]	$O(nm^2 \log n \log(n^2/m))$
Goldberg and Tarjan [1988]	$O(nm^2 \log^2 n)$
Orlin [1988]	$O((n+m') \log n S(n,m))$

題は最小費用流問題の特別な場合であり, さらにこれらの問題は線形計画問題に含まれている. このため, 次の関連研究に関するセクションでは最小費用流問題に関するアルゴリズムをあわせて紹介する.

1.2 関連研究

最小費用流問題はネットワークフロー理論の中でももっとも基礎的な問題の一つである. これまで研究者たちは理論的, または実践的な改良を行い, 今日までに多くのアルゴリズムが考案されている. 最小費用流問題における多項式時間アルゴリズムは 2 つに分類することができる. すなわち, 強多項式時間アルゴリズムと（弱）多項式時間アルゴリズムである.

強多項式時間アルゴリズムとは,

1. 数学的演算の数が入力サイズ（本研究では入力はグラフであるから, そのノードの数およびそのエッジの数）の多項式で抑えられる.
2. アルゴリズムの実行において, 比較, または加減算のみを演算として用いる.

上の二つの条件を満たしたアルゴリズムをいう.

最初の（弱）多項式時間アルゴリズムは 1972 年に Edmonds と Karp によって開発された. その後,1980 年に Rock が,1985 年に Bland と Jensen が,1987 年,1988 年に Goldberg と Tarjan が,1988 年に Ahuja,Goldberg,Orlin,Tarjan の 4 人がそれぞれ多項式時間アルゴリズムを開発した.

最初の強多項式時間アルゴリズムは 1985 に Tardos によって開発された. その後, 多くの強多項式時間アルゴリズムが開発されている. 強多項式時間で最小費用流問題を解くアルゴリズムとして [3][7][8][9] 等が知られている. これらをまとめたものを, 表 1.1 に示す. ただし,m を入力されるグラフの辺の数,n を同様に点の数とする. また,m' は有限の上界をもつ辺の数を表している.

ヒッチコック型輸送問題が最小費用流問題の特別な場合であることはすでに述べた. この

ため, ヒッチコック型輸送問題は最小費用流問題へと容易に変換することができる. 最小費用流問題を解く最良のアルゴリズムとしてOrlinによるアルゴリズム [3] が知られている. このアルゴリズムでの計算時間は$N = n+k$,$M = kn$とすると$O(N \log N(M+N \log N))$である. これを用いてヒッチコック型輸送問題を解けば, その計算時間は$O(kn^2 \log n+n^2 \log^2 n)$となる. ただし,$n \geq k$ であるとする.

現実に供給点の数nが需要点の数kに比べてたいへん大きいということはしばしば起こる. kを定数と見なせるとき,nについて線形時間でヒッチコック型輸送問題を解くアルゴリズム [10] も考案されている. しかし, このアルゴリズムの計算時間は$O(n(k!)^2)$であり,kが極端に小さくなければとても大きくなる. また, このような偏りのある二部グラフ（すなわち,$k \ll n$である場合）における, ネットワークフローアルゴリズムを応用し, ヒッチコック型輸送問題を$O(k^2 n \log(nC))$時間で解くアルゴリズム [11] も考案されている（Cはエッジに割り当てられたコストの最大値）. ほかにも, 幾何学的な解釈を与えることでOrlin のアルゴリズムを$O(k^2 n \log^2 n)$時間に改良する方法や確率的なアルゴリズムで計算時間を$O(nk + (c^2 n^2 k^{10} \log^7 n)^{\frac{1}{3}})$に抑えるアルゴリズム [12] が提案されている.

1.3 提案手法の目的と概要

現実の問題においては, 最小費用流問題の最適解ではなくその解における総費用が知りたいことも多い. この費用が特定の値以内に収まるかどうかを判定するためである. 前のセクションで示した例では, 決められた通信コストでデータの記録が行えるかどうかを判定したい場合がこれにあたる.

また, 画像の検索に関する分野では二つの画像間の距離を定義するために,Earth Mover's Distance[1]（以下,EMD と表記する）と呼ばれる距離関数を用いることがある. EMD はユークリッド空間上の重みつき点集合間に距離を定義する. これは次の手順で利用される. まず, 画像を重みつきの点集合として表現する. 画像内の各画素を色空間上に配置することで, この空間上での度数分布を求める. これはカラーシグネチャと呼ばれる. この様子を図 1.3 に示す. つぎに, 一方の画像からなる点集合を供給点集合とみなし, 他方を需要点集合とみなす. これら二つの点集合からなる完全二部グラフをつくる. 任意の二点間をつなぐエッジのコストはその二点間の色空間上での距離とする. このように定義されたグラフについてヒッチコック型輸送問題を解く. 得られた最小の費用, すなわち最適解における総費用が二つの集合間の距離であるとしている. この様子を図 1.4 に示す. 画像検索においてはこの距離が設定された閾値以上であるかどうかが重要な情報である. 距離がどのような値であるかということや, 最適解がどのようなフローで構成されているかということはさほど重要ではない. これはヒッチコック型輸送問題での最適な費用が閾値を越えるか否かを判定する問題となる. しかし, このような判定問題に対する効率的な解法はあまり良く知られていない.

このため, 本論文では最小費用流問題の費用に対する判定問題を扱う. これまでに述べたように, このような判定問題に対する効率的なアルゴリズムの需要が存在すること, ま

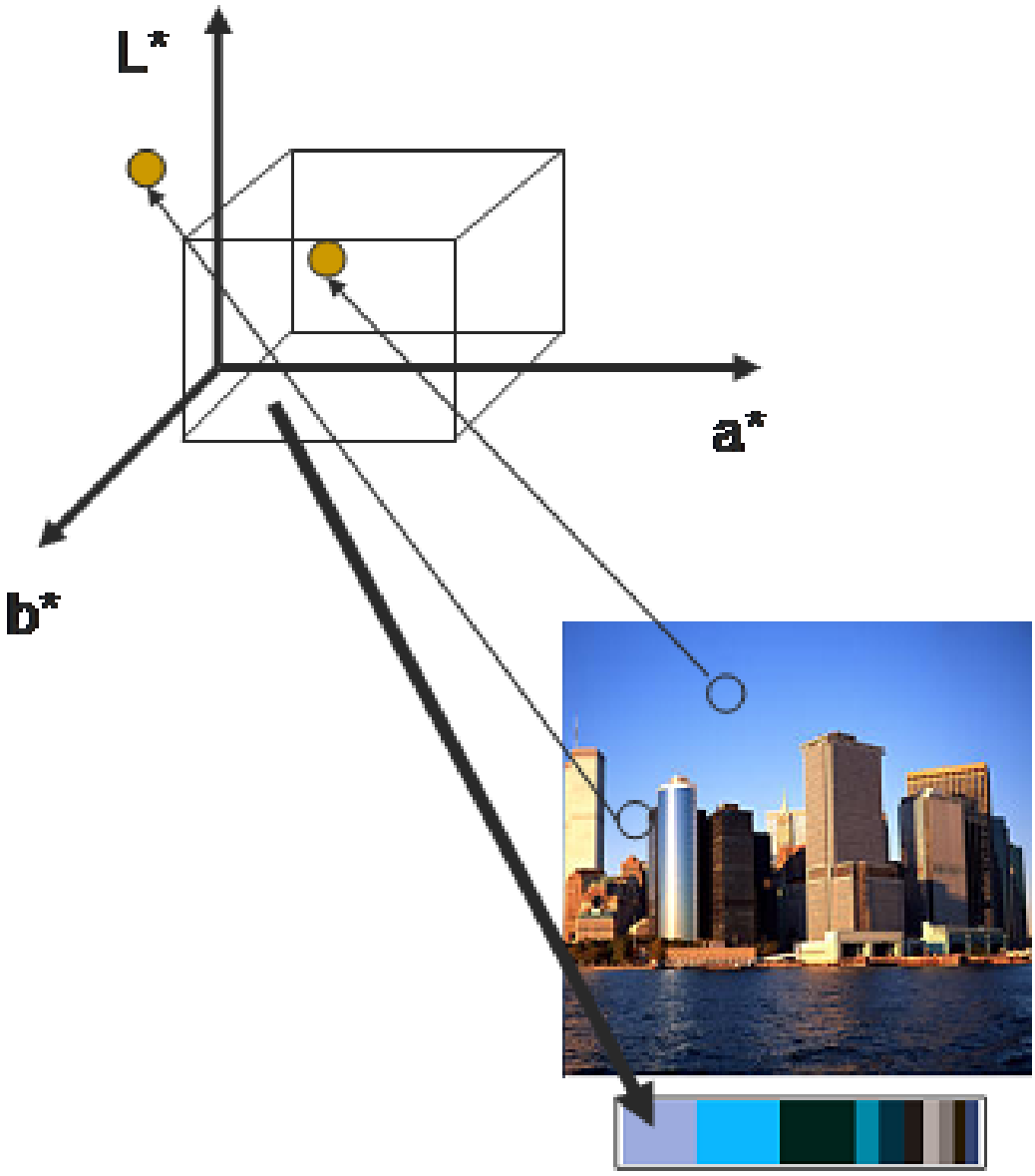

図 1.3: カラーシグネチャ

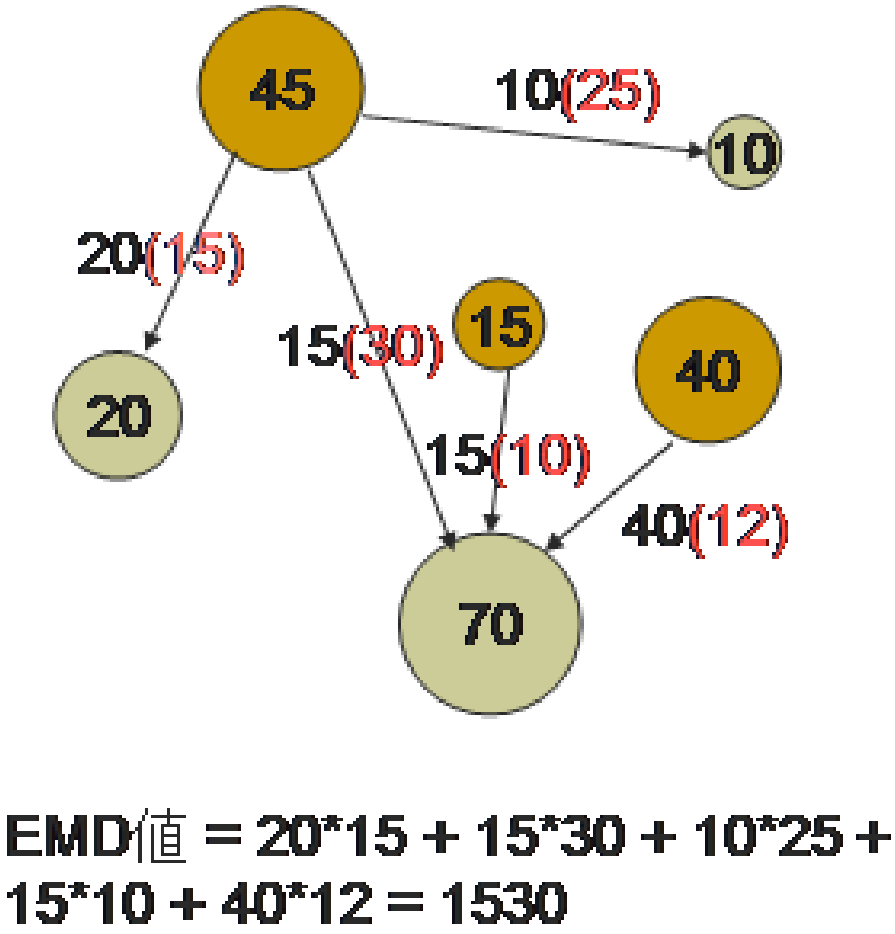

図 1.4: Earth Mover's Distance

た, このようなアルゴリズムが知られていないことが理由である.

1.4 本論文の構成

2 章では問題の定式化を行う. また, この問題に対する望ましい特性について議論する. 3 章では最小費用流問題を幾何学的性質を用いてより小さな最小費用流問題へと変換する手法を提案する. この結果得られた費用がもとの最小費用流問題の費用の下界を与えることを示す. 4 章では提案手法の計算量について述べる. 提案手法の最悪の計算量は単純な方法で解を求めてから判定する方法と同じであることを示す. 5 章では今回提案したアルゴリズムの実装を行った結果について述べる. 乱数を用いて発生させた問題に対して提案したアルゴリズムを適用し, 計算時間がどのように変化するかを示す. 6 章では今後の研究課題について触れ, 本論文のむすびとしている.

第2章　問題の定式化

ここでは本論文で扱う問題について定式化を行う. まず最初に最小費用流問題について定式化を行う. その後, 最小費用流問題にいくつかの条件を加え, ヒッチコック型輸送問題の定式化を行う. これらの定式化された問題から望ましい特性について述べ, 本研究の目的を述べる.

2.1　最小費用流問題

入力としてグラフ $G = (V, A)$, 各枝の上限容量 $\bar{c} : A \to R$, 下限容量 $\underline{c} : A \to R$, 単位流量当りの費用 $\gamma : A \to R$, および各頂点の需要または供給量 $b : V \to R$ が与えられる. ただし,R は実数とする. このとき, 容量制約

$$\underline{c}(a) \leq \varphi(a) \leq \bar{c}(a) \quad \forall a \in A$$

および流量保存則

$$\sum_{a \in \delta^+ v} \varphi(a) - \sum_{a \in \delta^- v} \varphi(a) = b(v) \quad \forall v \in V$$

を満たすような各枝の流量 $\varphi : A \to R$ の中で費用の総和

$$\sum_{a \in A} \gamma(a)\varphi(a)$$

を最小とする φ を出力する問題を最小費用流問題という. 流量 φ はフローともいい, 容量制約と流量保存則を満たす φ^* を実行可能流という. 費用を最小とするような実行可能流 φ^* を最小費用流という. この最小費用流を求める問題を最小費用流問題と呼ぶ.

2.2　ヒッチコック型輸送問題とその判定問題

与えられた有効グラフが 2 部グラフ $(V_1, V_2; A)$ であるとする. 需要および供給が

$$b(v) \geq 0 \quad (\forall v \in V_1) \quad ; \quad b(v) \leq 0 \quad (\forall v \in V_2)$$

であるとする. この条件に加え, 各枝について容量制約のない最小費用流問題をヒッチコック型輸送問題という. このようなグラフの例を図 2.1 に示す.

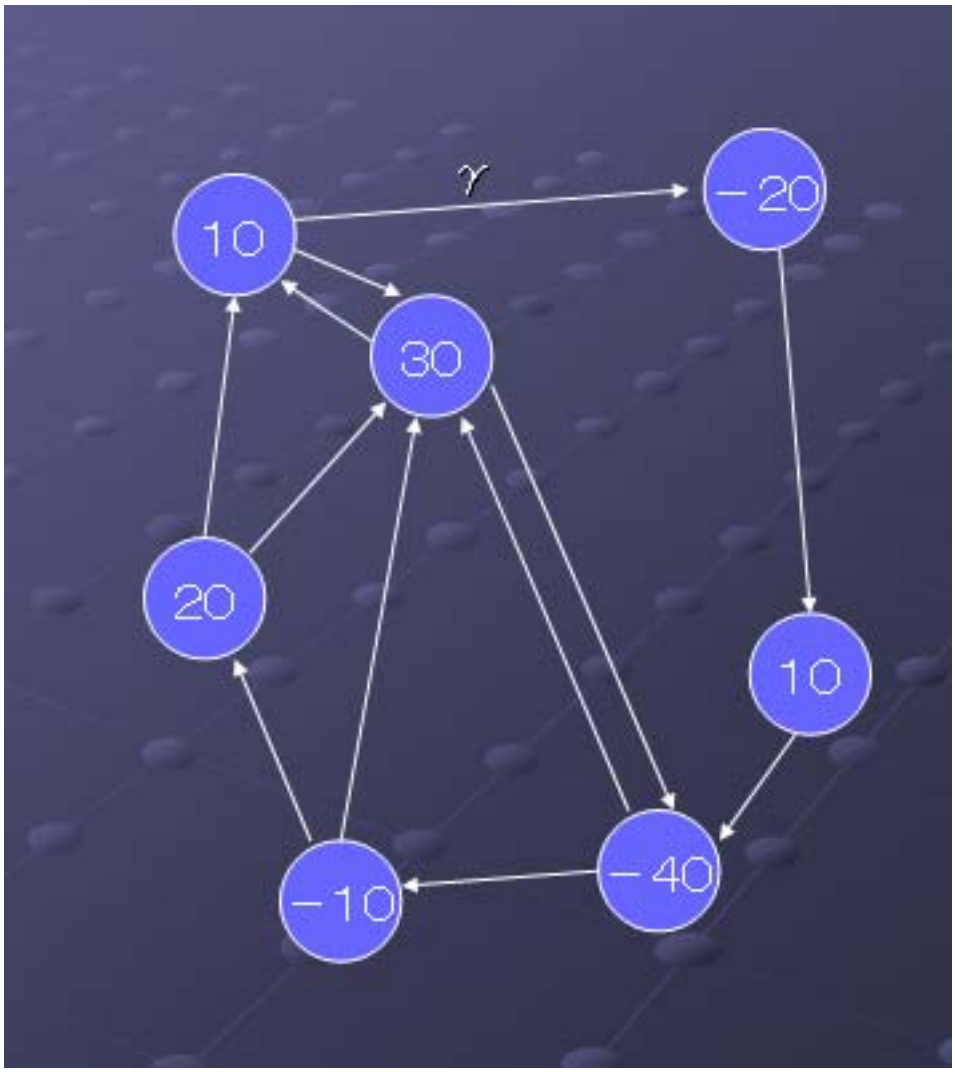

図 2.1: ヒッチコック型輸送問題の例

本論文で対象とする問題は, ヒッチコック型輸送問題における最小の総費用と設定された閾値との大小関係を判定することである. すなわち, ある閾値 X に対して

$$\sum_{a \in A} \gamma(a)\varphi^*(a) < X$$

が真であるかを判定する.

2.3 本研究の目的

このような判定問題とヒッチコック型輸送問題との計算複雑さの違いについてはあまり知られていない. また, 素朴な方法でこの判定問題を扱う場合を考えよう. このとき,

1. 入力されたグラフに対してヒッチコック型輸送問題を解く
2. 得られた解の総費用を閾値と比較する

といった手順を踏む. この方法では, 最適解の総費用と閾値との差に関係なく入力されたグラフの大きさに依存した一定の計算時間が必要であることは明らかであろう.

ここで EMD を例に考えよう. 図 2.2 に示すように明らかに違う画像の組と, 設定された閾値に近い距離を持つ画像の組を比較する. このとき, 二つの組の計算時間が等しい事実は望ましくないと考えられる.

そこで, 本論文では素朴なアルゴリズムを改良し, より短い計算時間でこの判定問題を解くアルゴリズムの提案を行う. 特に, このアルゴリズムは, 閾値と最適解の総費用が大きな差を持つ場合にはより短い計算時間で判定を行える. 具体的には, 入力されたグラフを縮約し, 小さなグラフから段階的にヒッチコック型輸送問題を解くことを繰り返す. 総費用と閾値との差が大きく, 前のセクションで述べた判定式が満たされないことが明らかならばその段階で計算を終了する. 判定に利用するグラフが元のグラフに比べて十分小さなうちに計算を終了できれば計算時間を短縮できる. また, 総費用と閾値との差が小さく, 元のグラフについてヒッチコック型輸送問題を解く必要がある場合でも, その計算量が単純に元のグラフについてヒッチコック型輸送問題を解いた場合と変わらないことを示すことができる. 一つの下界を算出する時間が最適解を求めるために必要な時間に比べ, 十分小さければ効率の良い手法といえる.

ただし, 計算の繰り返しにより計算時間が増加することが考えられる. 最悪の場合には与えられたグラフについてヒッチコック型輸送問題を解く必要が生じるために, このときには計算時間が増えてしまう. しかし, 後ほど述べるように提案手法での計算量は, 単純に最適解を求めた場合の計算量と同じになることがいえる.

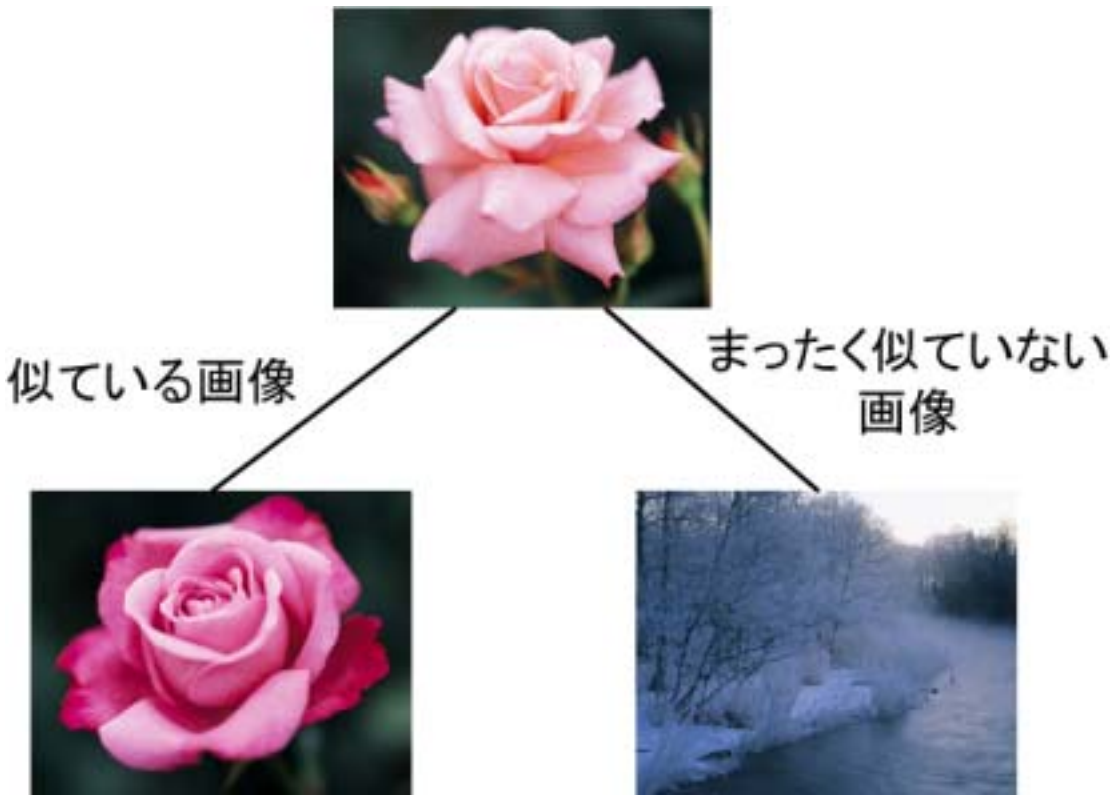

図 2.2: 似ている画像と全く似ていない画像

第3章　途中棄却を含む段階的解法

ここではヒッチコック型輸送問題に対して, 途中棄却を含む段階的解法を提案する. まず, 提案するアルゴリズムの概要について述べる. その後, このアルゴリズムの動作を述べる. このアルゴリズムは主に二つの部分に分けられるが, それぞれについて詳しく述べる. 次に, このアルゴリズムが正しく動作することを述べる. すなわち, このアルゴリズムによって棄却された問題では, 常に最適解の費用が閾値を超えていることを示す. 最後に, このアルゴリズムの計算時間について解析を行う. 提案したアルゴリズムは最悪の場合に素朴なアルゴリズムよりも多くの計算時間を必要とするが, 計算複雑さは等しいことを示す.

3.1　提案手法の概要

本研究では, 入力となるグラフに以下の事柄を仮定する.

1. 2 分グラフのすべての点はユークリッド空間上に配置されている.
2. 全ての辺は, その 2 点間の距離を単位当りのコストとしてもつ.

この仮定は, 実際の応用においてしばしば妥当である. これまでに述べた EMD の計算において, この仮定は成り立つ. なぜなら, 一般に利用される色空間では距離の三公理が成り立つからである. 輸送問題の名が表すような, ものを輸送する場合を考えよう. いくつかの工場でつくられた商品を全国のお店に届ける. この時移動のコストがその距離によって決まるとすれば, やはり, 上で示した仮定は成り立つ.

提案するアルゴリズムでは, まず, グラフの縮約を行う. ここでは幾何学的な性質を利用している. 縮約されたグラフに対して J.B.Orlin らによる最小費用流のアルゴリズム [3] を適用し, 閾値との比較を行う. 縮約されたグラフに対するヒッチコック型輸送問題の費用が, 縮約を行う前のグラフに対する下界を与えることから, 比較の結果, この費用が閾値を超えていることがわかれば, 縮約を行う前のグラフの費用も閾値を必ず超えることがいえる.

3.2　提案するアルゴリズム

ユークリッド空間上の供給点を表すベクトル集合を S, 需要点を表すベクトル集合を T とする. ただし, それぞれの集合の要素数を $|S| = n, |T| = k$ とし,$n \geq k$ を仮定する. 問題

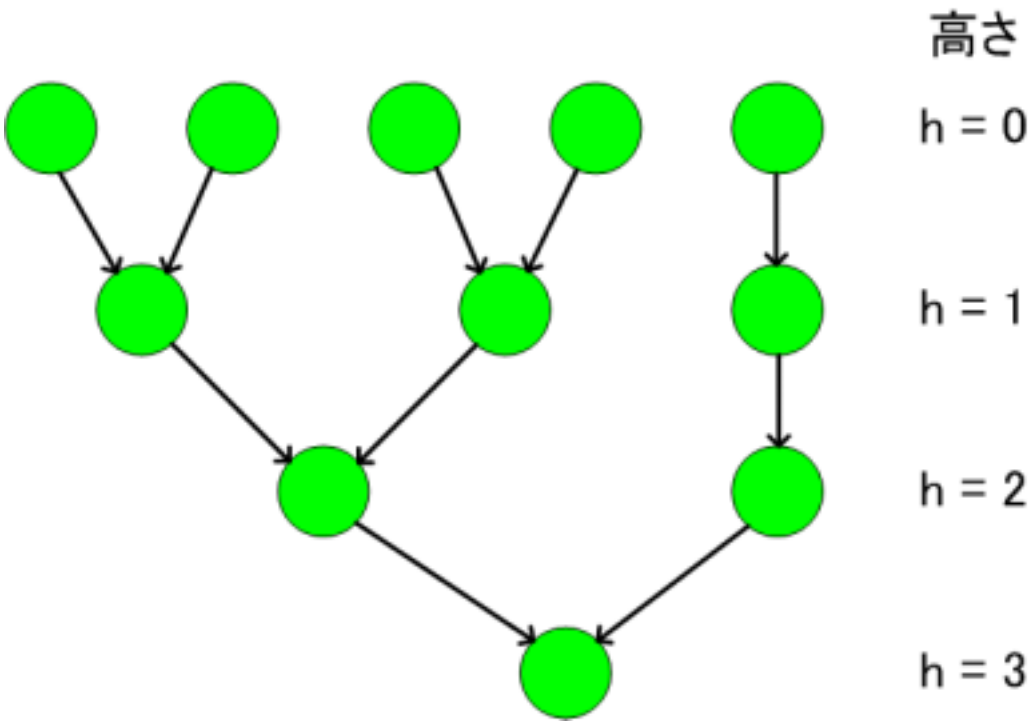

図 3.1: アルゴリズムが作成する木の例

の対象性からこの仮定により一般性が失われることはない. S（または T）の要素 s（または t）は $w : s \to R$ である重み w を持つ.

つぎに, 集合 S,T から木 S',T' を生成する. ただし, この木においては根の高さを 0 とし, 葉に向かってその高さが増加するものとする. この様子を図 3.1 に示す. 高さ h にある点集合を $S'(h), T'(h)$ とする. S',T' を以下のように生成する.

1. まず,$S'(L_s)$（または $T'(L_t)$）の要素は S（または T）の全要素とする. ただし,L_s（L_t）は木 S'（T'）の葉の高さである. すなわち,$L_s = \lceil \log |S| \rceil$ であり, 同様に,$L_t = \lceil \log |T| \rceil$ である.
2. 次の操作を $S'(L_s)$ から $S'(0)$ まで繰り返す. $S'(0)$ はただひとつの要素からなる点集合である.
   (a) $S'(h-1)$ から二つの要素 s_a, s_b を取り出し, その重心 s_c を求める. すなわち,$s_c = (w_a s_a + w_b s_b)/(w_a + w_b)$ である. さらに,$w(s_c) = w(s_a) + w(s_b)$ とし, これを $S'(h)$ の要素として加える.
   (b) これを $S'(h-1)$ の要素がなくなるまで繰り返す. ただし,$S'(h-1)$ の要素数が奇数ならば, 残ったひとつの点はそのまま $S'(h)$ の要素とする.

二つ目の操作の様子を図 3.2 に示す. 濃い色の点が薄い色の点に縮約されている. 点の大きさはその点が持つ重みを表している.

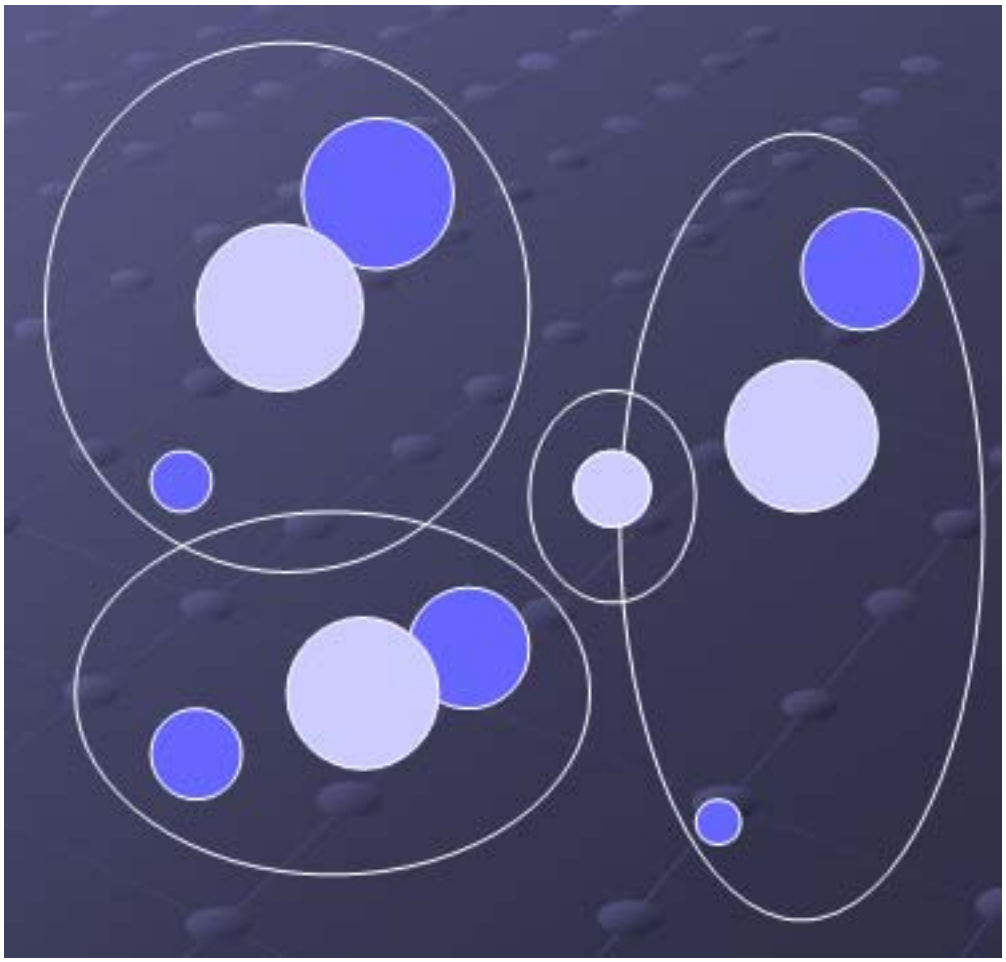

図 3.2: グラフ縮約の様子

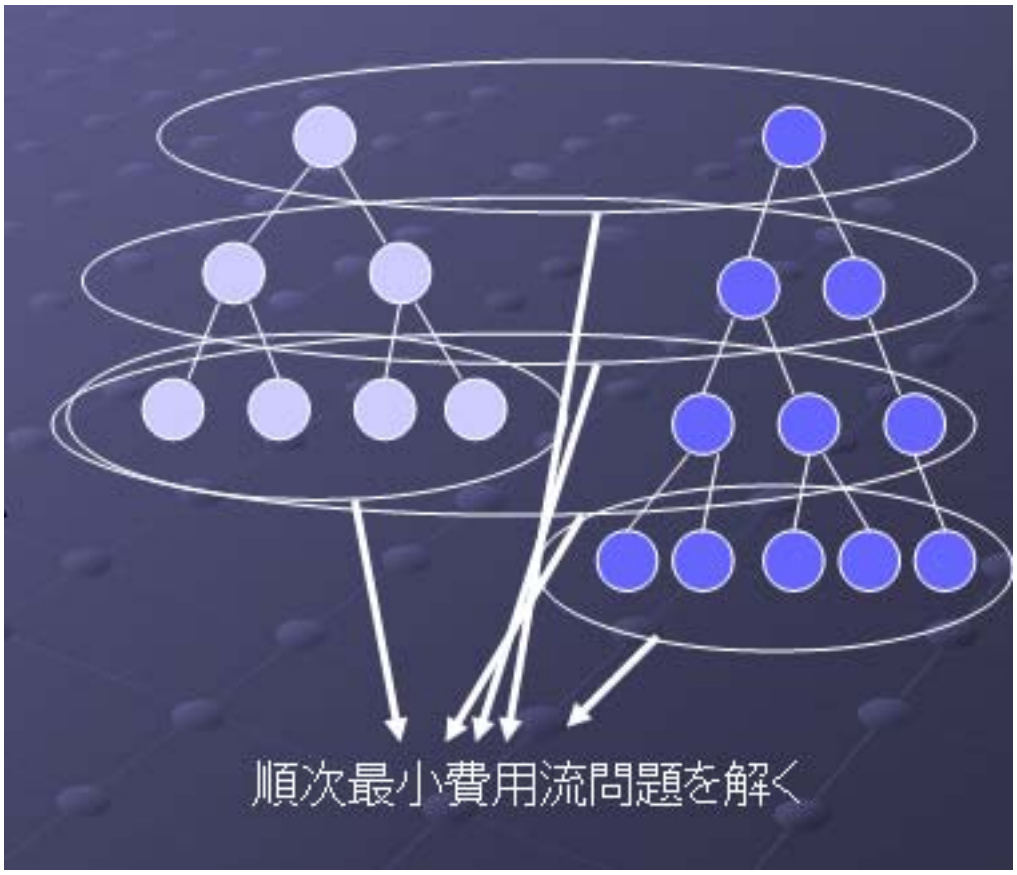

図 3.3: 段階的な実行の様子

これらの操作によりそれぞれ, 高さ $\lceil \log |S| \rceil, \lceil \log |T| \rceil$ である木が生成される. また, 高さ h に含まれる点の数はそれぞれ 2^h 個以下になることは生成される木が二分木であることから明らかである. この事実は, 後の計算時間の解析で用いる.

提案手法は, 高さ 0 から順に $\lceil \log n \rceil$ まで h を順次増加させながら, 各高さの点からなるグラフを作成する. このグラフに対し最小費用流問題を解く. この様子を図 3.3 に示す.

各高さにおける最小費用流問題の解がそれよりも大きな高さにおける解の下界であることを次に示す. 任意の高さにおけるグラフでの最適解の費用が, 閾値を超えた場合はその時点で計算を終了する. これは, 次のセクションで述べるように任意の高さにおけるグラフでの最適解の費用が, 閾値を超える場合, 元の入力されたグラフにおける費用は必ず閾値を超えることによる.

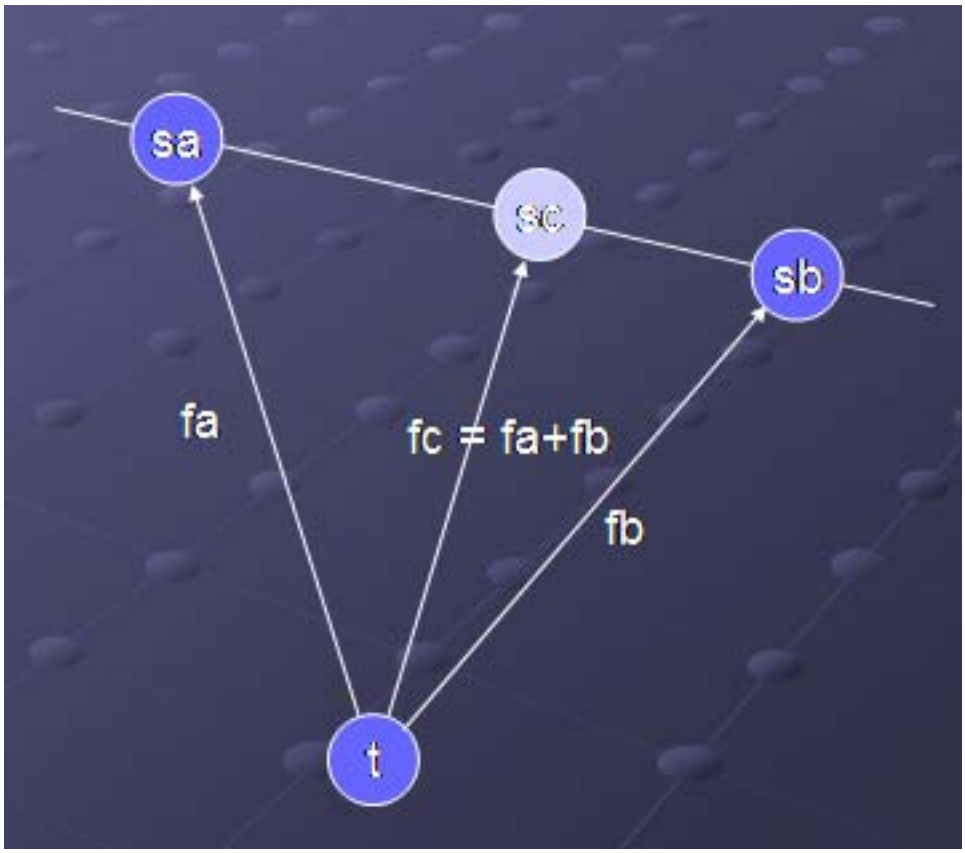

図 3.4: 提案手法でのフローの関係

3.3 アルゴリズムの正当性

s_a, s_b を $S'(h)$ の要素とし,$T'(h)$ の任意の要素を t とする. また,s_a と s_b の重心を $s_c \in S'(h+1)$ とする. いま, 最適なフローが与えられているとして,s_a から t に向かうフローの量を w_a とし, 同様に s_b から t へ向かうフローの量を w_b とする. このとき, 得られた重心から t へ向かうフローを $w_c = w_a + w_b$ とおく. これらを図 3.4 に示す. さらに,$\{s_1, s_2, \ldots\}$ から $T'(h)$ 全体へ流れるフローの費用を $C(s_1, s_2, \ldots)$ とすれば,

$$
\begin{aligned}
C(s_c) &= \sum_{t \in T'(h)} w_c |s_c - t| \\
&= \sum_{t \in T'(h)} (w_a + w_b) \left| \frac{w_a s_a + w_b s_b}{w_a + w_b} - t \right| \\
&= \sum_{t \in T'(h)} |w_a s_a + w_b s_b - (w_a t + w_b t)| \\
&= \sum_{t \in T'(h)} |w_a (s_a - t) + w_b (s_b - t)|
\end{aligned}
$$

となる. また,

$$C(s_a, s_b) = \sum_{t \in T'(h)} w_a|s_a - t| + w_b|s_b - t|$$

である. これらから,

$$\begin{aligned} C(s_a, s_b) - C(s_c) &= \sum_{t \in T'(h)} w_a|s_a - t| + w_b|s_b - t| \\ &\quad - \sum_{t \in T'(h)} |w_a(s_a - t) + w_b(s_b - t)| \\ &\geq 0 \end{aligned}$$

が成り立つ. 最後の不等号は,S, T がユークリッド空間上の点であり, 任意のスカラ a, b と任意のベクトル x, y に関して三角不等式 $a|x| + b|y| \geq |ax + by|$ が成り立つことによる.

また, 対象性から $T'(h+1)$ についても同様の議論が成り立つ.

この様子を図 3.5 に示す. この操作によって得られた $S'(h+1), T'(h+1)$ におけるフローは明らかに実行可能流である. このため, その総費用は高さ $h+1$ での解の上界を与えることがわかる. このことが任意の h について言えることから, 任意の高さにおける費用がそれよりも大きな高さで費用の下界を与えることがわかる.

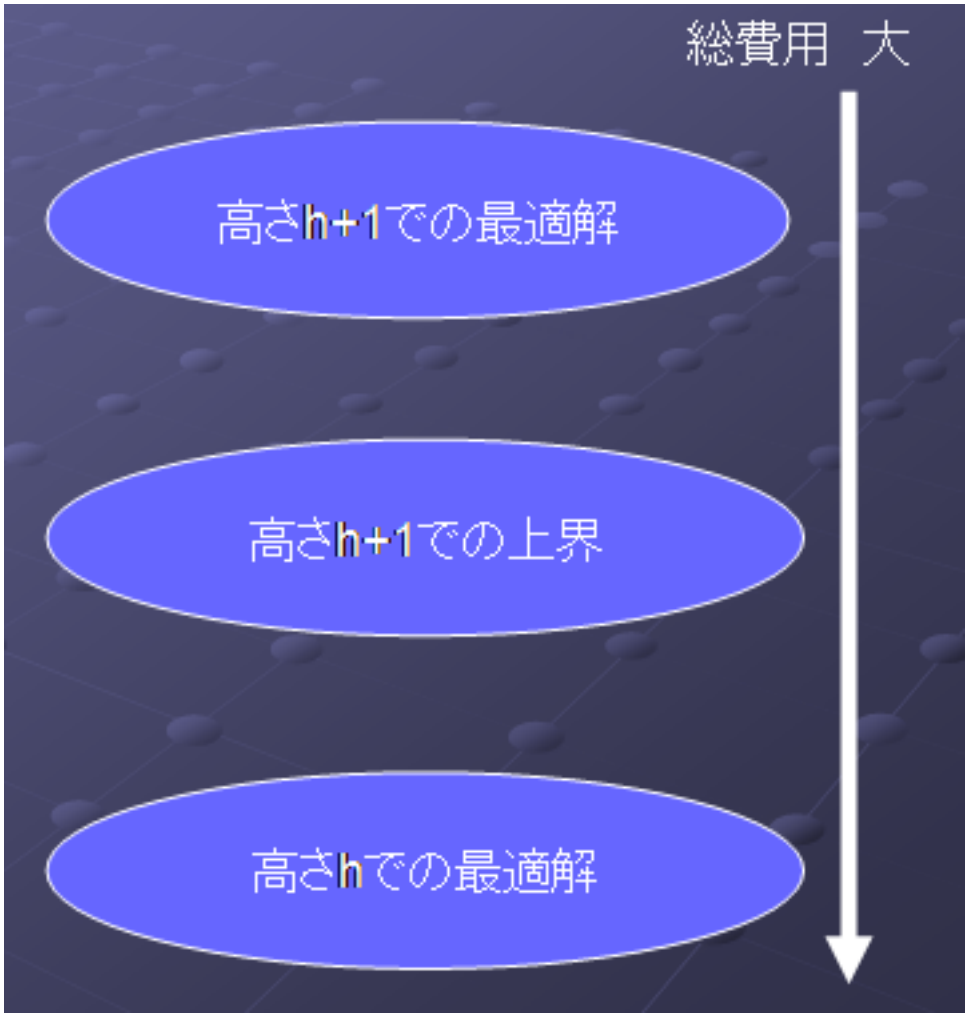

図 3.5: 費用の関係

第4章　提案手法の計算複雑さ

ここでは提案手法の計算量について論じる. 提案したアルゴリズムは大きく分けて二つの操作からなる. 一つは, 木の作成であり, もう一つは段階的な判定問題である. この結果, 提案したアルゴリズムの計算複雑さは素朴なアルゴリズムを超えないことが示せる.

4.1　計算複雑さの解析

まず, 木の作成に必要な計算量を考える. 一つの重心を作成するのに必要な時間は $O(1)$ であり, 木全体のノードの数は最初に与えられた点の数の二倍を越えないから全体で $O(n)$ となる.

次に段階的に最小費用流問題を解くのに必要な計算量を求める.

$$(2^0, 2^1, \ldots, 2^{\lceil \log \log n \rceil})$$

なる系列に沿って高さを増加させ, その高さの点集合からなるグラフについて最小費用流問題を解く. 高さ h には高々2^h 個の点が存在することは既に述べた.

よって, 計算量は,

$$Time(n,k) = \sum_{i=0}^{\lceil \log \log n \rceil} n(i)^2 k(i) \log n(i) + n(i)^2 \log^2 n(i)$$

となる. ただし,$n(i), k(i)$ はそれぞれ $S'(i), T'(i)$ の要素数とする.

これを解くと,

$$\begin{aligned}
Time(n,k) &= \sum_{i=0}^{\lceil \log \log n \rceil} n(i)^2 k(i) \log n(i) + n(i)^2 \log^2 n(i) \\
&\leq \sum_{i=0}^{\lceil \log \log n \rceil} n^2 k \log 2^{2^i} + n^2 \log^2 2^{2^i} \\
&= \sum_{i=0}^{\lceil \log \log n \rceil} n^2 k 2^i + n^2 (2^i)^2 \\
&\leq n^2 k 2^{\lceil \log \log n \rceil + 1} + n^2 \left(2^{\lceil \log \log n \rceil + 1}\right)^2 \\
&= O(n^2 k \log n + n^2 \log^2 n)
\end{aligned}$$

となる.

これは, 素朴なアルゴリズムの計算複雑さと同じである. 提案したアルゴリズムが最も多くの計算時間を必要とするのは, 木を根から葉まで辿り, 計算を続ける必要がある場合である. しかし, そのような場合でも元の問題を解くのと同じ計算複雑さを持つことがわかる.

第5章　提案したアルゴリズムの実装とその結果

ここでは提案アルゴリズムの実装とその実験結果について述べる．まず, 実験環境について述べる．主に使用した計算機と有用なライブラリである LEDA について述べる．その後, 実験結果について述べる．ここでは素朴なアルゴリズムと提案したアルゴリズムの計算時間について比較を行った．最後に得られた実験結果から考察を行う．

5.1　実験環境

考案したアルゴリズムの実装を行った．以下の環境で実験を行った．LEDA はドイツのマックスプランク研究所で開発された C++のクラスライブラリである．効果的なアルゴリズムと豊富なデータ構造を提供している．

5.2　実験結果

乱数を用いて 10 個から 640 個の点集合を順次生成し, これを入力とした．それぞれ, 点の数を固定して 100 回の試行を行った．総費用が閾値を越え, 棄却される割合である棄却率が 50%, または 95% 程度となるように閾値を設定した．それぞれについて点の個数毎に平均の実行時間と標準偏差を求めた．これを用いて, 実験結果をエラーバー表記を用いて表した．全ての場合について素朴なアルゴリズムと提案したアルゴリズム両方で実験を行った．

表 5.1: 実験環境

使用した言語	C++
用いたライブラリ	LEDA
計算機	Sun Fire15K
CPU	Sun UltraSPARC-III+ (900MHz)
OS	Solaris8

点の数	計算時間の平均	分散	標準偏差	割合
10	0.00267	0	0	1.9071
20	0.0081	0	0	0.9878
40	0.0322	0.00022	0.0148	0.8895
80	0.1505	0.00368	0.060663	0.8699
160	0.765	0.153	0.3911	0.9063
320	4.762	5.1582	2.2764	0.8227
640	99.748	1031.85	32.122	0.942

表 5.2: 提案手法の計算時間（棄却率 50%）

点の数	計算時間の平均	分散	標準偏差
10	0.0014	0	0
20	0.0082	0	0
40	0.0362	0	0
80	0.173	0.000191	0.01382
160	0.844	0.00449	0.067
320	4.762	0.881	0.9386
640	105.883	419.391	20.479

表 5.3: 従来手法の計算時間（棄却率 50%）

提案手法での実験結果を表 5.2 に示す. 各列はそれぞれ入力される点の個数, 判定に要した時間の平均, その分散, および標準偏差, そして従来手法に対する計算時間の割合を表している. 分散と標準偏差の項目で 0 であるものは時間が短すぎて計測できなかったことを表している.

従来手法を用いて行った実験結果を表 5.3 に示す. 各列はそれぞれ入力される点の個数, 判定に要した時間の平均, その分散, および標準偏差を表している.

エラーバーを用いてグラフにしたものをそれぞれ図 5.1, 図 5.2 に示す.

これらからわかるように, 棄却率 50% では提案手法と従来手法には計算時間の差が見られない. また, 提案手法の従来手法に対する計算時間の割合も, 点の数が増減してもさほど変化しない.

次に棄却率を 95% としたときの結果を表 5.4, 表 5.5 に示す.

同じく, これらをエラーバーを用いて表したものを図 5.3, 図 5.4 に示す.

これらからわかるように, 棄却率が高くなると提案手法は従来手法に比べ少ない計算時間で判定を終えることができる. また, 点の数が多くなるほど計算時間の割合は小さくなっていくことがわかる.

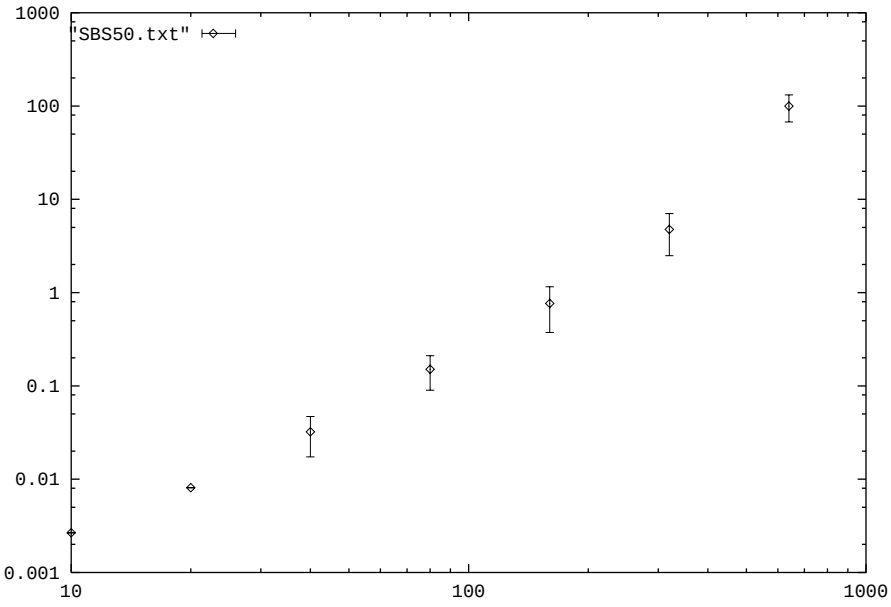

図 5.1: 提案手法の計算時間（棄却率 50%）

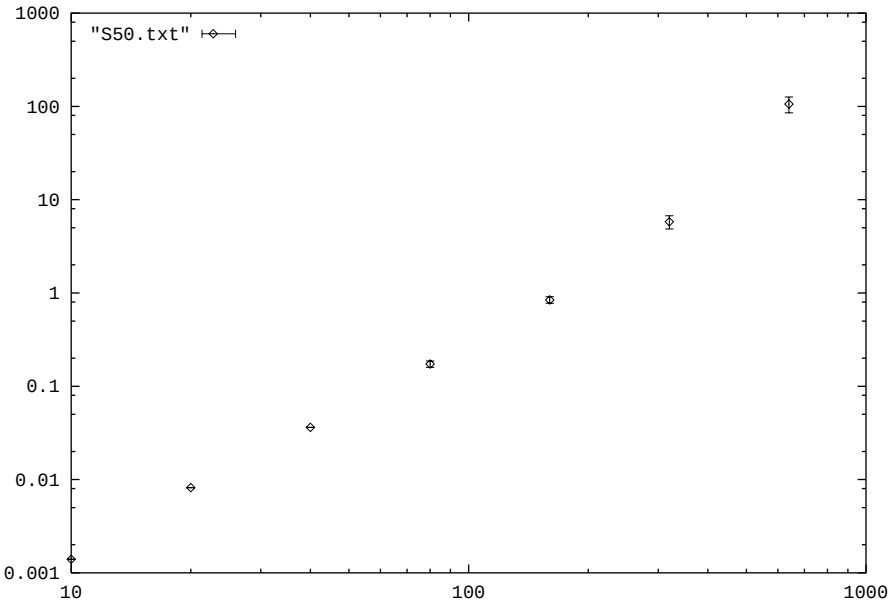

図 5.2: 従来手法の計算時間（棄却率 50%）

点の数	計算時間の平均	分散	標準偏差	割合
10	0.0014	0	0	0.6363
20	0.00549	0	0	0.6614
40	0.0298	0.00027	0.0164	0.8075
80	0.1469	0.00385	0.06204	0.8332
160	0.1557	0.00418	0.06465	0.1896
320	0.2089	0.00431	0.06565	0.03291
640	72.537	3102.98	55.704	0.6402

表 5.4: 提案手法の計算時間（棄却率 95%）

点の数	計算時間の平均	分散	標準偏差
10	0.0022	0	0
20	0.0083	0	0
40	0.0369	0	0
80	0.1763	0.000239	0.01546
160	0.821	0.0056	0.074833
320	6.345	1.0259	1.01286
640	113.289	452.522	21.272

表 5.5: 従来手法の計算時間（棄却率 95%）

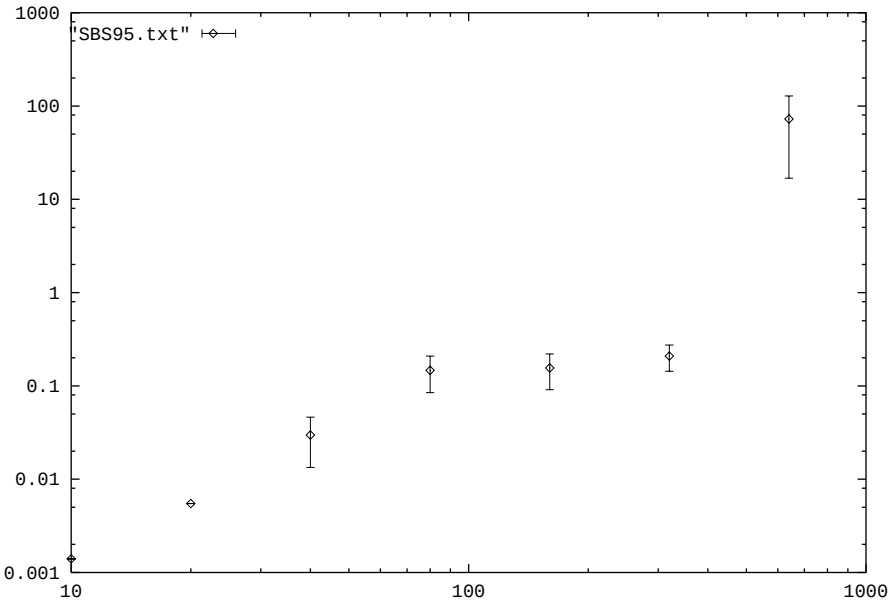

図 5.3: 提案手法の計算時間（棄却率 95%）

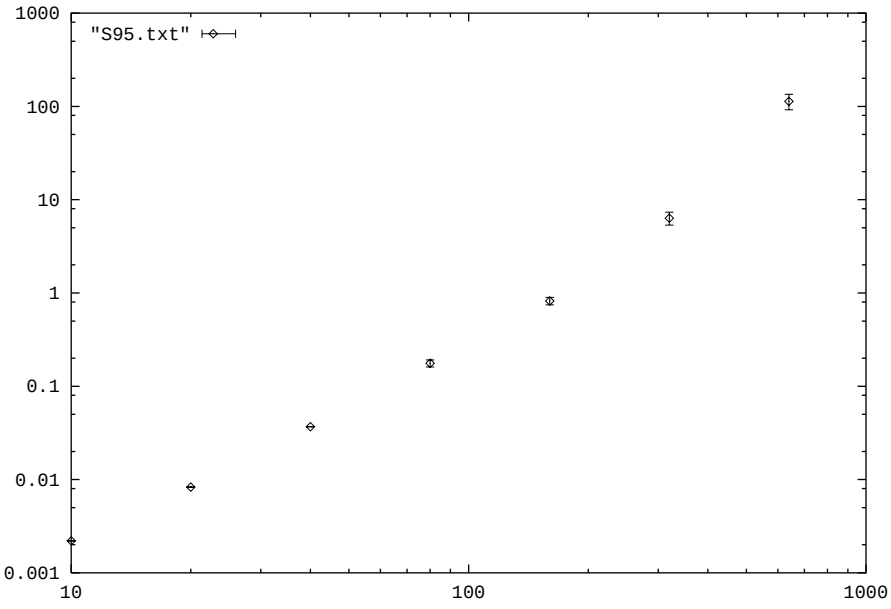

図 5.4: 従来手法の計算時間（棄却率 95%）

5.3 実験結果に対する考察

棄却率が 50% の場合には従来手法と提案手法に差が見られなかった. これは, 次のように考えられる. たとえば棄却率が 50% であれば, 提案手法では棄却されなかった分の計算時間が従来手法の 50% 分は少なくともあることになる. さらに提案手法ではこのような場合に余分な計算を行っている. このため, これらの計算時間が支配的になるため, 従来手法と提案手法には差が見られなかったと考えられる.

棄却率が 95% の場合には, 従来手法と提案手法との間には明確な差が見られる. 点の数が増加するほど, この差は明確になる. 点の数が多いほど, 早い段階で棄却された場合の利益は大きくなる. このため, 棄却率が高い場合は点の数が多いほど, 提案手法は大きな利益を得られるからであると考えられる.

これらの結果から, 棄却率, 点の数がともに大きな場合には, 提案手法は従来手法に比べ計算時間の減少が期待できることがわかる. ところで, 画像の検索等の分野では棄却率がとても大きいことが知られている. このため, 提案手法はこのような分野で有用であると考えられる.

第6章　むすび

本研究では, ヒッチコック型輸送問題に対する判定問題について効果的なアルゴリズムの提案を行った. 実験の結果から, 提案したアルゴリズムは有用な特徴を持つことがわかった. また, 素朴なアルゴリズムに対して計算複雑さが変化しないこと保証した. 提案手法の利点は, 閾値と大きく異なる費用を解としてもつ場合には早く計算を終了することができ, 閾値と近い値の場合には時間がかかるといった, 我々の感覚に近い動作をすることがあげられる. 一般に, データ数が増加するほど棄却されるべきデータ数も増加し, 棄却率は高くなる. このため, 提案手法は実用的であると考えられる. また, 新たな最小費用流問題のアルゴリズムが考案された場合も容易に流用可能であることも利点としてあげることができる.

今後の研究課題として以下のことを考えている. ひとつ目は, 棄却率の変化に対する計算時間の増減についての解析である. 任意の棄却率に対して, 計算時間の期待値を求めることは可能であると考えている. また, 今回はユークリッド空間上の点集合である場合を取り扱った. そうでない場合には下界を得ることは容易ではないと思われるが, この問題についても考えてみたい. さらに, データベースの検索等への応用も今後の課題であると考えている.

第7章　謝辞

本研究の全過程を通じ終始懇切丁寧なる御指導, 御鞭撻を賜った 北陸先端科学技術大学院大学情報基礎学講座 浅野 哲夫 教授に, 衷心より厚く御礼申し上げる.

筆者が本研究を行うにあたり, 細部にわたり熱心かつ適正なる御指導, 御助言を頂いた, 情報基礎学講座の 中野 浩嗣 助教授, 小保方 幸次 助手, 元木 光雄 助手に心より深謝申し上げる.

さらに, 本学大学院生の 寺本 幸生 氏, 千葉 英史 氏には熱心な御討論を頂いた.

その他, 浅野研究室ならびに中野研究室の諸氏には種々の面でお世話になった. ここに心から感謝の意を表する.

ここに記して, 以上の方々に深甚なる感謝の意を捧げる.

関連図書

[1] Y. Rubner, L. Guibas, C. Tomasi. The Earth Mover's Distance, Multi-Dimensional Scaling, and Color-Based Image Retrieval. In *Proceedings of DARPA Image Understanding Workshop*, pp. 661-668, 1997.

[2] T. Tokuyama, J. Nakano. Efficient Algorithm for the Hitchcock Transportation Problem. In *SIAM Journal on Conputing Volume 24, Number 3*, pp. 563-578, 1995.

[3] J. B. Orlin. A FASTER STRONGLY POLYNOMIAL MINIMUM COST FLOW ALGORITHM. In *Proceedings of the 20th ACM Symposium on Theory of Computing*, ACM, pp. 377-387, 1988.

[4] M. Charikar, R. Panigrahy. Clustering to Minimize the Sum of Cluster Diameters. In *Proceedings of 33rd Annual ACM Symposium on Theory of Computing*, 2001.

[5] Y. Bartal, M. Charikar, D. Raz. Approximating min-sum k-Clustering in Metric Spaces. In *Proceedings of 33rd Annual ACM Symposium on Theory of Computing*, 2001.

[6] T.Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A. Y. Wu. A Local Search Approximation Algorithm for k-Means Clustering. In *Proceedings of the 18th Annual ACM Symposium on Computational Geometry*, June 5-7, 2002.

[7] S.Fujishige An $O(m^3 \log n)$ Capacity-rounding Algorithm for the Minimum Cost Circulation Problem: A Dual Framework to Tardos's Algorithm, In *Math. Prog. 35*, pp.298-309, 1986.

[8] Z.Galil, E.Tardos An $O(n^2(m + n \log n) \log n)$ Min-cost Flow Algorithm, In *Proc, 27th IEEE FOCS*, pp.136-146, 1986.

[9] E.Tardos A Strongly Polynomial Minimum Cost Circulation Algorithm, In *Combinatorica 5*, pp.247-255, 1985.

[10] T.Matsui Linear Time Algorithm for Hitchcock Transportation problem with Fixed Number of Supply Points, *Preprint*, 1991.

[11] R.K.Ahuja, J.B.Orlin, C.Stein, R.E.Tarjan Improved Algorithms for Bipartite Network Flow, *Preprint*, 1991.

[12] T.Tokuyama, J.Nakano Efficient Algorithms for the Hitchcock Transportation Problem, In *SIAM J. Computing 24-3*, pp.563-578, 1995.