

青 山 学 院 大 学 理 工 学 部  
経 営 シ ス テ ム 工 学 科

卒 業 論 文

多目的最短経路問題のための  
拡張ベルマンフォード法の提案

2 0 1 8 年 度

中野 壱帥

15715051

指導教員

宋 少秋

印

# 目次

第1章	はじめに	3
1.1	研究背景	3
1.2	研究目的	4
1.3	章構成	4
第2章	諸定義	5
2.1	多目的最適化問題	5
2.2	パレート最適解	5
2.3	最短経路問題	6
2.4	多目的最短経路問題	10
第3章	従来研究	12
3.1	拡張ダイクストラ法	12
3.2	完全多項式時間近似スキームによる解法	13
3.3	ラベル付けアルゴリズムによる解法	16
第4章	解法の提案と実験的評価	18
4.1	問題の分析	18
4.2	非負の問題に対する解法の提案	24
4.2.1	提案解法1	25
4.2.2	提案解法2	30
4.3	負の重みを考慮した問題に対する解法の提案	34
4.4	実装における工夫	37
4.5	提案解法の評価と分析	43
第5章	結論	50
5.1	研究成果	50

5.2 今後の課題 . . . . .	51
参考文献	51
付 録 A 多目的最短経路問題に対する解法の実験結果	53

# 第1章 はじめに

## 1.1 研究背景

現代には道路ネットワークや通信ネットワークなど様々なネットワークが存在する。これらのネットワークに対する最適化を行う場合、複数の目的関数を考慮する場合がある。一般に複数の目的関数値を最大化（最小化）する問題は多目的最適化問題と呼ばれている。多目的最適化問題のインスタンスは最適化目的の数が  $n$  である場合、 $n$  個の実数値関数または整数値関数  $f: A \rightarrow \mathbb{R}$  で定義される。すなわち、 $\vec{f} = (f_1, \dots, f_n)$  と表される。最小化問題の場合、 $\min f(\vec{x})$  となる  $x$  を求める。多目的最適化問題の場合、それぞれの目的関数がトレードオフの関係となる場合がある。トレードオフとは一方を追求すれば他方を犠牲にせざるを得ない関係であり、同時に最適にできない。あらかじめ各目的関数に対する比重を決めて解を求めると1つの解が適切に求めることができる。しかし、探索者の各目的関数に対する比重が決まっていない場合や複数の探索者がいる場合、求まった1つの解がそれぞれの探索者に対して適切な解ではない場合がある。そこで、パレート最適解の集合を求める。

パレート最適解とは取りうる値の範囲を全て考慮した上で支配されない解である。最適化目的の数を  $k$ ，目的関数を  $f$  とし、解  $x, y$  が  $\forall i \in \{1, \dots, k\}, f_i(x) \leq f_i(y) \wedge \exists i \in \{1, \dots, k\}, f_i(x) < f_i(y)$  を満たすとき、 $x$  は  $y$  を支配する。全ての目的関数を同時に最適化できる解が存在する場合、その解はパレート最適解となる。また、その解と等価な解のみパレート最適解となる。パレート最適解の集合を求めることによりそれぞれの探索者が最適解となり得る解の集合から解を選択できるので、解を選択する意思決定を容易にできる。多目的最短経路問題ではそれぞれの目的関数は辺の重みとして与えられ、経路の評価はその経路上の辺の重みの合計である。一般にパレート最適解は指数的に存在するため、列挙する数に対して効率的な解法を求めたい。

単一目的最短経路問題には負の重みを考慮した解法が提案されているが、調査した限り従来の多目的最短経路問題には負の重みを考慮した研究がなされていない。単一目的最短経路問題において負の重みを考慮した場合、負のサイクルが存在し解が求められない場

合がある．しかし，多目的最短経路問題において負の重みを考慮した場合，1つの目的関数において負のサイクルが存在しても他の目的関数に対する解を求められる場合がある．そこで，負の重みを含む多目的最短経路問題において負のサイクルが存在しない目的関数に対する効率的な解法を考える．

## 1.2 研究目的

従来研究では非負の重みに対する解法が提案されている．本研究では負の重みを許す解法を提案するため，ダイクストラ法ではなくベルマンフォード法を基にした解法を考える．多目的の場合，格納されるデータは経路である．単一目的ベルマンフォード法では単に全ての辺を緩めるため，全ての経路を更新対象とし，探索順序の指定がされていない．同じ経路を何度も更新しても新しい経路は発見されないため無駄が生じる．また，従来研究で提案されている解法は同じ経路を複数回更新したり，全ての経路から必要とする経路を探索したりしているため，効率的なデータ構造が必要である．探索順序の指定がされていないと無駄な経路の更新を行なってしない無駄が生じることが多い．そのため，データ構造の設計によりそれぞれの問題を解決した解法を提案する．目的関数によって負のサイクルを持つことが考えられ，その検出法を含む解法を提案する．

## 1.3 章構成

本論文の章構成は以下である．

- 第2章では，多目的最適化問題と最短経路問題に対する定義を紹介し，多目的最短経路問題の解に対する定義を行う．
- 第3章では，多目的最短経路問題に対する従来研究を紹介する．
- 第4章では，提案解法の紹介と実装における工夫，本研究に対する成果を述べる．
- 第5章では，結論として本研究の成果と今後の課題について述べる．

## 第2章 諸定義

この章では，多目的最適化問題と最短経路問題に対する定義を紹介する．多目的最適化問題における解となるパレート解の説明をする．単一目的最短経路問題に対する解法を紹介する．また，多目的最短経路問題の解に対する定義を行う．本研究における定式化をする．(負の重みを考慮した場合を含む)

### 2.1 多目的最適化問題

最適化問題とは与えられたインスタンスに対して実行可能な最適解を求める問題である．最適化問題のインスタンスは特定の集合上で定義された実数値関数または整数値関数  $f: A \rightarrow \mathbb{R}$  で定義される．最小化問題の場合，  $\min f(x)$  となる  $x$  を求める．すなわち，  $x_0 \in A: \forall x \in A, f(x_0) \leq f(x)$  となる  $x_0$  を求める．最大化問題の場合，  $\max f(x)$  となる  $x$  を求める．

多目的最適化問題とは複数の目的関数に対する最適化問題である．多目的最適化問題のインスタンスは最適化目的の数が  $n$  である場合，  $n$  個の実数値関数または整数値関数  $f: A \rightarrow \mathbb{R}$  で定義される．すなわち，  $\vec{f} = (f_1, \dots, f_n)$  と表される．最小化問題の場合，  $\min \vec{f}(x)$  となる  $x$  を求める．多目的最適化問題の場合，それぞれの目的関数がトレードオフの関係にある場合が存在し，全ての目的関数値が最大（最小）となる最適解が存在するとは限らない．(例：  $f_1(x_0) < f_2(x_0) \wedge f_1(x_1) > f_2(x_1)$  ) 一般的に，多目的最適化問題はパレート最適解の集合を求める．パレート最適解は複数の目的関数をそのまま考慮された解なので，求めたい選好解を見つけることや挙動変数の関係を知ることが可能になる．多目的最適化問題には Konaka ら [1] が提案した遺伝的アルゴリズムなどがある．

### 2.2 パレート最適解

パレート最適解は，多目的最短経路問題における解の支配関係により定義される．解  $x, y$  が以下の条件を満たすとき，  $x$  は  $y$  を支配する．

- $\forall i \in \{1, \dots, k\}, f_i(x) \leq f_i(y)$
- $\exists i \in \{1, \dots, k\}, f_i(x) < f_i(y)$
- $k$  : 最適化目的の数,  $f$  : 目的関数

パレート最適解とは取りうる値の範囲を全て考慮した上で支配されない解である。

## 2.3 最短経路問題

有向もしくは無向グラフをグラフ  $G = (V, E)$  と表す。グラフは頂点集合  $V = \{v_1, \dots, v_n\}$  , 辺集合  $E = \{e_1, \dots, e_m\}$  から成る。重み付きグラフの場合, 全ての辺  $e \in E$  は重み  $e_w$  を持つ。無向グラフにおいて, 両端点を  $s, t$  とする辺  $e \in E$  を  $s, t \in e$  と表す。有向グラフにおいて, 始点を  $s$  , 終点を  $t$  とする辺  $e \in E$  を  $e = (s, t)$  と表す。  $s$  から  $t$  までの距離を  $d_{s,t}$  と表す。最短経路問題とは重み付きグラフ  $G = (V, E)$  の与えられた2つのノード  $s, t$  間を結ぶ経路の中で, 重みが最小の経路 ( $\min d_{s,t}$  となる経路) を求める最適化問題である。

### 最短経路問題の種類

- 2 頂点对最短経路問題 : 特定の 2 つのノード間の最短経路問題。

入力 : 重み付きグラフ, 始点  $s$  , 終点  $t$

出力 :  $s$  から  $t$  への最短経路

- 単一始点最短経路問題 : 特定の 1 つのノードから他の全ノードとの間の最短経路問題。

入力 : 重み付きグラフ, 始点  $s$

出力 :  $s$  から全頂点への最短経路

- 全点对最短経路問題 : グラフ内のあらゆる 2 ノードの組み合わせについての最短経路問題。

入力 : 重み付きグラフ

出力 : 全頂点間の最短経路

### 最短経路問題の主な解法

- 幅優先探索

始点から近い順に探索する．重みがない（すべての重みが1である）最短経路問題に使われる．通った辺の本数に応じて重みが決まるため探索によって発見した頂点は最短経路が決定する．計算時間は  $O(E)$  である．以下に無向グラフにおける単一始点最短経路問題の解法を示す．

入力：グラフ  $G = (V, E)$ ，始点  $s \in V$ ， $Q \leftarrow \emptyset$

出力： $s$  から全ての頂点への経路

Step 1.  $Q \leftarrow s$

Step 2.  $Q = \{\emptyset\}$  になるまで以下の操作を行う．

Step 2-1.  $Q$  の先頭にあるノード  $v$  を取り出す．

Step 2-2.  $u = \{u \in V \mid v, u \in e \wedge e \in E\}$  が未探索のとき， $Q \leftarrow u$  ．

Step 3. 経路を出力する．

2 頂点对最短経路問題の場合，終点が見つかった時点で探索を終了し，始点から終点への経路を出力する．入力が有向グラフの場合：

Step 2-2.  $u = \{u \in V \mid e = (v, u) \wedge e \in E\}$  が未探索のとき， $Q \leftarrow u$  ．

- ダイクストラ法

全ての重みが非負であるグラフについての有名なアルゴリズムであるダイクストラ法について説明をする．ダイクストラ法は Dijkstra E W[9] によって開発された．ダイクストラ法は全ての重みが非負であるグラフにおいて使われる．探索手順はすでに探索済みのノードの中で重みが最小のノードを求め，更新対象として探索していく．全ての重みが非負の場合，探索したノードの中で重みが最小のものはその後の探索で更新されることはないので重みが決定する．ダイクストラ法の計算時間は  $O(V^2)$  である．以下に無向グラフにおける単一始点最短経路問題の解法を示す．

入力：グラフ  $G = (V, E)$ ，始点  $s \in V$ ， $E$  の各辺の長さ， $Q \leftarrow \emptyset$

出力： $s$  から全ての頂点への経路

Step 1.  $s_w = 0$  とし， $v \in V/\{s\}$  に対して  $v_w = \infty$  とする．

Step 2.  $Q \leftarrow s$



**Step 3.**  $Q = \{\emptyset\}$  になるまで以下の操作を行う。

**Step 3-1.**  $v = \{v \in Q \mid v_w \leq v'_w \wedge v' \in Q\}$  を取り出す。

**Step 3-2.**  $e = \{e \in E \mid v \in e\}$  について以下の操作を行う。

**Step 3-2-1.**  $u = \{u \in V \mid u \in e\}$  となる  $u$  に対して,  $u_w > v_w + e_w$  を満たすとき以下の操作を行う。

**Step 3-2-1-1.**  $u_w = v_w + e_w$  .

**Step 3-2-1-2.**  $u \notin Q$  のとき,  $Q \leftarrow u$  .

**Step 4.** 経路を出力する。

2 頂点对最短経路問題の場合, 終点が更新対象となった時点で探索を終了し, 始点から終点への経路を出力する。入力が有向グラフの場合:

**Step 3-2.**  $u = \{u \in V \mid (v, u) = e \wedge e \in E\}$  となる  $u$  に対して,  $u_w > v_w + e_w$  を満たすとき以下の操作を行う。

**Step 3-2-1.**  $u_w = v_w + e_w$  .

**Step 3-2-2.**  $u \notin Q$  のとき,  $Q \leftarrow u$  .

- ベルマンフォード法

グラフ内の全ての重みが非負であるかどうかに関わらず使用できる有名なアルゴリズムであるベルマンフォード法について説明をする。ベルマンフォード法は Bellman Rら [7] によって開発された。全ての重みの実数であるグラフにおいて使われる。(負の重みを含んでも使用できる) ベルマンフォード法の探索順序は頂点数を  $|V|$  とした時, 全辺を緩めることを単に  $|V| - 1$  回繰り返す。辺  $e$  を緩めるとは,  $e$  に接続しているノードの重みを  $v_w, u_w$  とし,  $e$  の重みを  $e_w$  としたとき,  $u_w$  が  $v_w + e_w$  によって更新されることである。 $|V| - 1$  回の探索終了時にもう一度全辺を緩め, 更新が行われた場合は負の閉路が存在するため負の閉路の存在を報告する。負の閉路が存在しない場合は経路の長さの最大が全頂点を通る  $|V| - 1$  となる。負の閉路が存在する場合は無限に更新が行われるため,  $|V|$  回目の探索でも更新が行われる。よって,  $|V|$  回目の探索により負の閉路の存在有無を確かめられる。ベルマンフォード法の計算時間は  $O(EV)$  である。以下に無向グラフにおける単一始点最短経路問題の解法を示す。

入力: グラフ  $G = (V, E)$ , 始点  $s \in V$ ,  $E$  の各辺の長さ

出力：  $s$  から全ての頂点への経路，負の閉路の存在有無

**Step 1.**  $s_w = 0$  とし，  $v \in V/\{s\}$  に対して  $v_w = \infty$  とする．

**Step 2.**  $|V| - 1$  回以下の操作を行う．

**Step 2-1.**  $e = e \in E \mid v, u \in e$  となる  $e$  に対して，  $u_w > v_w + e_w$  を満たすとき以下の操作を行う．

**Step 2-1-1.**  $u_w = v_w + e_w$  ．

**Step 3.** Step 2-1 を行いノードの重みが更新された場合，負の閉路の存在を報告する．

**Step 4.** 経路を出力する．

2 頂点对最短経路問題の場合，単一起点最短経路問題を解き，始点から終点への経路を出力する．入力が有向グラフの場合：

**Step 2-1.**  $e = e \in E \mid (v, u) = e$  となる  $e$  に対して，  $u_w > v_w + e_w$  を満たすとき以下の操作を行う．

**Step 2-1-1.**  $u_w = v_w + e_w$  ．

- ワーシャルフロイド法

グラフ内の全ての重みが非負であるかどうかに関わらないかつ全点对最短経路をもとめる際に使用できる有名なアルゴリズムであるワーシャルフロイド法について説明をする．ワーシャルフロイド法は R. W. Floyd ら [5] によって開発された．ワーシャルフロイド法は重み付き有向グラフにおいて全点对最短経路を多項式時間で解くアルゴリズムである．3つの頂点  $a, b, c$  を選んで， $a \rightarrow b \rightarrow c$  という道が  $a \rightarrow c$  という道より短ければ  $a \rightarrow c$  の距離を更新するという操作を全ての頂点の組み合わせで繰り返して最短距離を確定させていく． $a, b, c$  はそれぞれ  $|V|$  の選択が出来るので， $a \rightarrow b \rightarrow c$  という道が  $a \rightarrow c$  という道より短ければ  $a \rightarrow c$  の距離を更新するという操作は  $|V|^3$  回行われる． $|V|^3$  回の操作後，更新できる解が存在する場合は負の閉路が存在する．ワーシャルフロイド方は負の重みにも対応でき，計算時間は  $O(V^3)$  である．以下に全点对最短経路問題の解法を示す．

入力： グラフ  $G = (V, E)$ ，始点  $s \in V$ ， $E$  の各辺の長さ

出力： 全頂点对の経路

**Step 1.**  $s_w = 0$  とし,  $v \in V/\{s\}$  に対して  $v_w = \infty$  とする.

**Step 2.** 各  $1 < k < |V|$  に対して以下の操作を行う.

**Step 2-1.** 各  $1 < i < |V|$  に対して以下の操作を行う.

**Step 2-1-1.** 各  $1 < j < |V|$  に対して以下の操作を行う.

**Step 2-1-1-1.**  $d_{i,j} > d_{i,k} + dk, j$  を満たすとき,  $d_{i,j} = d_{i,k} + dk, j$

**Step 3.** 経路を出力する.

2 頂点間の距離がマイナスの場合, 負の閉路が存在する.

上記で説明した解法以外にも Ahuja ら [2] が提案した二分ヒープを用いたダイクストラや Thorup[3] が提案したコンポーネントとそのハイパーキーを応用する線形時間アルゴリズムなど最短経路問題に対する解法は数多く存在する.

## 2.4 多目的最短経路問題

多目的最短経路問題とは, 最短経路問題の最適化目的の数を複数にすることによって, 多目的最適化問題に拡張した問題である. つまり, 目的値が複数の重み付きグラフにおいて与えられた 2 つのノード間を結ぶ経路の中で, パレート最適解となる経路を求める最適化問題である. 有向もしくは無向グラフをグラフ  $G = (V, E)$  と表す. グラフは頂点集合  $V = \{v_1, \dots, v_n\}$ , 辺集合  $E = \{e_1, \dots, e_m\}$  から成る. 最適化目的の数を  $k$  とするとき, 全ての辺  $e \in E$  は重み  $\vec{e}_w = \{e_{w1}, \dots, e_{wk}\}$  を持つ. 無向グラフにおいて, 両端点を  $s, t$  とする辺  $e \in E$  を  $s, t \in e$  と表す. 有向グラフにおいて, 始点を  $s$ , 終点を  $t$  とする辺  $e \in E$  を  $e = (s, t)$  と表す.  $s$  から  $t$  までの距離を  $\vec{d}_{s,t} = \{d_{s,t1}, \dots, d_{s,t,k}\}$  と表す. また, 本研究では全ての値が同じとなる経路は 1 つのみ求める.

### 多目的最短経路問題の種類

- 多目的 2 頂点对最短経路問題: 特定の 2 つのノード間の最短経路問題.

入力: 重み付きグラフ, 始点  $s$ , 終点  $t$ , 最適化目的の数  $k$

出力:  $s$  から  $t$  へのパレート最適解となる経路の集合

- 多目的単一始点最短経路問題: 特定の 1 つのノードから他の全ノードとの間の最短経路問題.

入力：重み付きグラフ，始点  $s$ ，最適化目的の数  $k$

出力：  $s$  から全頂点へのパレート最適解となる経路の集合

- 多目的全点对最短経路問題：グラフ内のあらゆる 2 ノードの組み合わせについての最短経路問題.

入力：重み付きグラフ，最適化目的の数  $k$

出力：全頂点間のパレート最適解となる経路の集合

### 負の重みを含む場合

単目的最短経路問題では負の閉路が存在する場合，負の閉路の存在を報告し最短経路は求めなかった．これは負の閉路を何度も通過することによって重みを更新し続けるためである．しかし，多目的最短経路問題の場合，目的関数が複数存在するので 1 つの目的関数において負の閉路が存在する場合でも，他の目的関数による最適化をすることで解を求めることができる．よって，本研究では負の閉路が存在する目的関数を考慮しない解を求める．

## 第3章 従来研究

この章では多目的最短経路問題に対する従来研究の成果を紹介する.

入力: グラフ  $G = (V, E)$ , 始点  $s \in V$ , 最適化目的の数  $k$ , 各辺の重みを返す関数  $w : E \rightarrow \mathbb{R}^k$

出力:  $s$  から全ての頂点への最短経路となるパレート解の集合

記号

$k$ : 最適化目的の数

$v \in V$ ,  $j = 1, \dots, k$  に対して

$l_{v(j)}$ : 始点からノード  $v$  に到達したときに生じる第  $j$  番目の目的関数における総コスト

$e \in E$ ,  $j = 1, \dots, k$  に対して

$e_{(j)}$ : 辺  $e$  の第  $j$  番目の目的関数におけるコスト

### 3.1 拡張ダイクストラ法

従来研究として N.Takahashi ら [6] に提案された拡張ダイクストラ法について説明をする. 拡張ダイクストラ法の探索順序は探索済みの頂点  $v$  を指定し,  $v$  に隣接する頂点  $u$  に対する解を更新していく方法である.

また, いくつかの工夫によって探索空間を削除することが可能である. 拡張ダイクストラ法は全ての値が非負の場合のみ使用できる. 以下に無向グラフに対するアルゴリズムを記載する.

記号

$v \in V$  に対して

$W_v$  : ノード  $v$  に隣接するノード集合

アルゴリズム

**Step 1.**  $L_v \leftarrow (s, 0, \dots, 0)$ ,  $W_v \leftarrow \emptyset$ ,  $v \leftarrow s$

**Step 2.**  $W_v$  を求める.

**Step 3.**  $W_v$  内のノード全てに対して以下の操作を行う.

**Step 3-1.**  $\omega \in W_v$  を選択する.

**Step 3-2.**  $(v, l_{v(1)}, \dots, l_{v(k)}) \in L_v$  を選択する.

**Step 3-3.** 始点が  $v$ , 終点が  $\omega$  である辺  $e$  の重みベクトルを  $\vec{e} = (e_{(1)}, \dots, e_{(k)})$

とし,  $(\omega, l_{\omega(1)}^*, \dots, l_{\omega(k)}^*) \leftarrow (\omega, l_{v(1)} + e_{(1)}, \dots, l_{v(k)} + e_{(k)})$  とする.

**Step 3-4.**  $L_v$  と  $(\omega, l_{\omega(1)}^*, \dots, l_{\omega(k)}^*)$  に対してパレート最適解の判定を行う.

**Step 3-5.** 全ての頂点  $v \in V$  が探索されていないとき,  $v' \in W_v$  を選択し,  $v \leftarrow v'$  として Step 2 に戻る.

**Step 4.** 全てのパレート解を出力

パレート最適解の判定

**Step 1.**  $L_v$  と  $(\omega, l_{\omega(1)}^*, \dots, l_{\omega(k)}^*)$  を受け取る.

**Step 2.**  $L_v$  の全てのラベルに対して以下の操作を行う.

**Step 2-1.**  $(\omega, l'_{\omega(1)}, \dots, l'_{\omega(k)}) \in L_v$  を選択する.

**Step 2-2.** 以下の条件を満たすとき,  $L_v$  を返す.

- $\forall i \in \{1, \dots, k\}, l'_{\omega(i)} \leq l_{\omega(i)}^*$

**Step 2-3.** 以下の条件を満たすとき,  $L_v \leftarrow L_v \setminus \{(\omega, l'_{\omega(1)}, \dots, l'_{\omega(k)})\}$ .

- $\forall i \in \{1, \dots, k\}, l_{\omega(i)}^* \leq l'_{\omega(i)}$

- $\exists i \in \{1, \dots, k\}, l_{\omega(i)}^* < l'_{\omega(i)}$

**Step 3.**  $L_v \leftarrow L_v \cup \{(\omega, l_{\omega(1)}^*, \dots, l_{\omega(k)}^*)\}$  を返す.

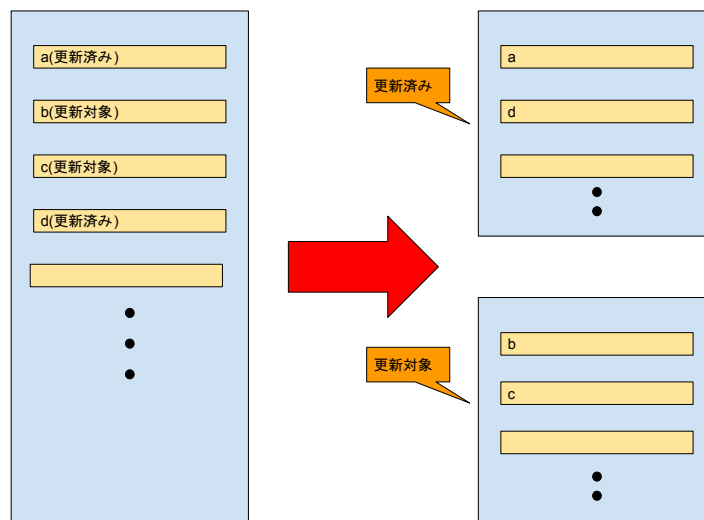
## 3.2 完全多項式時間近似スキームによる解法

従来研究として Thomas Breugem ら [4] に提案された完全多項式時間近似スキームによる解法について説明をする. 完全多項式時間近似スキームとは, 入力サイズが  $n$ , 精度

が  $1/\epsilon (\epsilon > 0)$  となる多項式時間アルゴリズムである。また、任意の整数  $\epsilon > 0$  に対して  $\alpha = 1 + \epsilon$  とできる入力サイズの多項式時間アルゴリズムを多項式時間近似スキームという。従来は全ての経路に対して更新を行っていたが更新しなくても良い経路があるため無駄が生じてしまっている。また、単目的最短経路問題と違い更新する際の比較対象が多く存在することから1回の更新でもより多くの時間や作業が必要となるため無駄な更新を避けることによって大きな効率化に繋がると予想できる。完全多項式時間近似スキームによる解法はラベルを2つ用意し、それぞれ以下とする。

- パレート解となり得るか更新対象としない経路のラベル
- パレート解となり得るか更新対象とする経路のラベル

図 3.1: 完全多項式時間近似スキームによる解法と従来とのラベル比較



完全多項式時間近似スキームによる解法は更新対象とする経路のラベルと更新対象としない経路のラベルを用意することによってすでに更新した経路を更新する無駄を削減し効率化することを目的としたアルゴリズムである。しかし、2つのラベルを用意するため多少メモリを多く使う。また、頂点毎の更新ではなく発見された経路から順に更新されていくため、更新順序も異なる。更新順序が異なると従来では一度解になり後の更新で削除されるはずだった経路が解とならずに探索が進んでいく場合や、従来では解にならなかったものが一度解として記憶される場合があるため探索中に更新する経路の数や削除される経路の数異なる。探索中に更新する経路の数や削除される経路の数異なることによって実装時間が異なる場合もある。更新対象とするラベルと更新対象としないラベルを分け

ることによってすでに更新した経路を再度更新する無駄がなくなるため実装時間は短くなる.

以下に無向グラフに対するアルゴリズムを記載する.

## 記号

$L_T$  : 更新対象とする経路のラベル

$L_P$  : 更新対象としない経路のラベル

## アルゴリズム

**Step 1.**  $L_T \leftarrow (s, 0, \dots, 0)$  ,  $L_P \leftarrow \emptyset$

**Step 2.**  $L_T = \{\emptyset\}$  となるまで以下の操作を行う.

**Step 2-1.**  $L_T$  の先頭にある  $(\omega, l_{\omega(1)}, \dots, l_{\omega(k)}) \in L_T$  を選択する.

**Step 2-2.**  $L_T \leftarrow L_T \setminus \{(\omega, l_{\omega(1)}, \dots, l_{\omega(k)})\}$

**Step 2-3.** 以下の条件を満たすとき,  $L_P \leftarrow L_P \cup \{(\omega, l_{\omega(1)}, \dots, l_{\omega(k)})\}$  とする.

- 任意の  $(\omega^*, l_{\omega^*(1)}, \dots, l_{\omega^*(k)}) \in L_P$  に  $(\omega, l_{\omega(1)}, \dots, l_{\omega(k)})$  が支配されない.
- 任意の  $(\omega^*, l_{\omega^*(1)}, \dots, l_{\omega^*(k)}) \in L_P$  と  $(\omega, l_{\omega(1)}, \dots, l_{\omega(k)})$  における全ての目的関数値が同値でない.

**Step 2-4.** 頂点  $\omega$  に辺  $e$  によって接続している頂点  $u$  に対して以下の操作を行う.

**Step 2-4-1.** 辺  $e$  の重みベクトルを  $\vec{e} = (e_{(1)}, \dots, e_{(k)})$  とし,  $(\omega', l_{\omega'(1)}, \dots, l_{\omega'(k)}) \leftarrow (\omega', l_{\omega(1)} + e_{(1)}, \dots, l_{\omega(k)} + e_{(k)})$  とする.

**Step 2-4-2.** 以下の条件を満たすとき,  $L_T \leftarrow L_T \cup \{(\omega', l_{\omega'(1)}, \dots, l_{\omega'(k)})\}$  とする.

- 任意の  $(\omega^*, l_{\omega^*(1)}, \dots, l_{\omega^*(k)}) \in L_P \cup L_T$  に  $(\omega', l_{\omega'(1)}, \dots, l_{\omega'(k)})$  が支配されない.
- 任意の  $(\omega^*, l_{\omega^*(1)}, \dots, l_{\omega^*(k)}) \in L_P \cup L_T$  と  $(\omega', l_{\omega'(1)}, \dots, l_{\omega'(k)})$  における全ての目的関数値が同値でない.



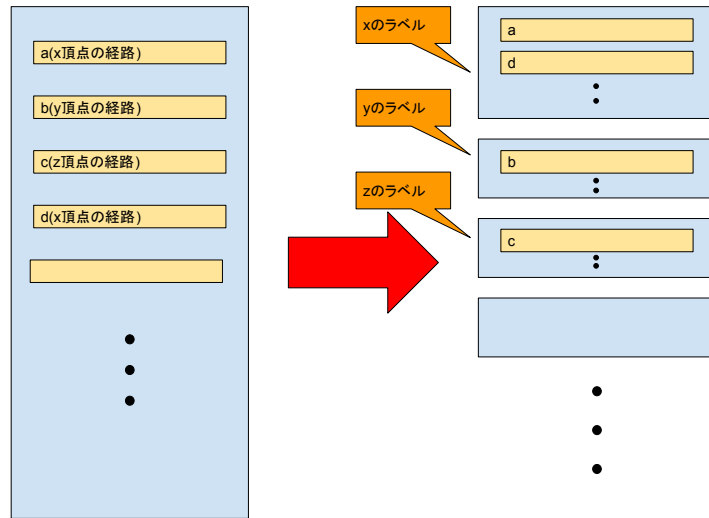
**Step 2-4-3.** 任意の  $(\omega'', l_{\omega''(1)}, \dots, l_{\omega''(k)}) \in L_T$  に対して  $(\omega', l_{\omega'(1)}, \dots, l_{\omega'(k)})$  が  $(\omega'', l_{\omega''(1)}, \dots, l_{\omega''(k)})$  を支配しているとき,  $L_T \leftarrow L_T \setminus \{(\omega'', l_{\omega''(1)}, \dots, l_{\omega''(k)})\}$  とする.

**Step 3.** 全てのパレート解を出力

### 3.3 ラベル付けアルゴリズムによる解法

従来研究として Jos Luis E. Dos Santos ら [8] に提案されたラベル付けアルゴリズムによる解法について説明をする. ラベル付けアルゴリズムとはいくつかのラベルを用意し, 経路を分けて記憶することにより条件毎の経路を参照しやすくするアルゴリズムである. ラベル付けアルゴリズムによる解法では頂点数だけラベルを用意し, 各頂点毎の経路をそれぞれのラベルに記憶していく.

図 3.2: ラベル付けアルゴリズムによる解法と従来とのラベル比較



従来のアルゴリズムでは1つのラベルから更新対象となる頂点に対する経路を探索し経路の更新を行っていたが, 1つのラベルには全ての頂点に対する経路が存在するため多くの経路に対して更新対象とするかの判定を行わなければならない. ラベル付けアルゴリズムによる解法により各頂点毎の経路を分けて保存しておくことが可能なので更新対象となる頂点に対する経路を選択するとき探索を行う必要がなくなり効率化になると考えられる. ラベル付けアルゴリズムによる解法の探索方法はベルマンフォード法を改良したもので, 未更新の頂点集合を用意し, 未更新の頂点がなくなるまで全辺を緩めて更新する. ベルマンフォード法のように全ての経路に対して更新を行ってしまうと無駄な更新

が多く行われてしまうため未更新の頂点集合（前回の探索によって発見された解に対する頂点集合）を用意することによって無駄な更新を削減している。

以下に無向グラフに対するアルゴリズムを記載する。

## 記号

$v \in V$  に対して

$L_v$  : 頂点  $v$  に対する経路のラベル

$X$  : 更新対象とする頂点集合

## アルゴリズム

**Step 1.**  $\forall v \in V, L_v \leftarrow \emptyset, L_s \leftarrow (s, 0, \dots, 0), X \leftarrow s$

**Step 2.**  $X = \{\emptyset\}$  となるまで以下の操作を行う。

**Step 2-1.**  $v \in X$  となる頂点  $v$  を選択する。

**Step 2-2.**  $X \leftarrow X \setminus \{v\}$

**Step 2-3.**  $e = e \in E \mid v, u \in e$  となる  $e$  に対して以下の操作を行う。

**Step 2-3-1.** 頂点  $v$  に対する全ての経路  $(v', l_{v'(1)}, \dots, l_{v'(k)}) \in L_v$  に対して以下の操作を行う。

**Step 2-3-1-1.** 辺  $e$  の重みベクトルを  $\vec{e} = (e_{(1)}, \dots, e_{(k)})$  とし,  
 $(u', l_{u'(1)}, \dots, l_{u'(k)}) \leftarrow (u', l_{v'(1)} + e_{(1)}, \dots, l_{v'(k)} + e_{(k)})$  とする。

**Step 2-3-1-2.** 以下の条件を満たすとき,  $L_u \leftarrow L_u \cup \{(u', l_{u'(1)}, \dots, l_{u'(k)})\}$ ,  
 $X \leftarrow X \cup \{u\}$  とする。

- 任意の  $(u^*, l_{u^*(1)}, \dots, l_{u^*(k)}) \in L_u$  に  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  が支配されない。
- 任意の  $(u^*, l_{u^*(1)}, \dots, l_{u^*(k)}) \in L_u$  と  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  における全ての目的関数値が同値でない。

**Step 2-3-1-3.** 任意の  $(u'', l_{u''(1)}, \dots, l_{u''(k)}) \in L_u$  に対して  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  が  $(u'', l_{u''(1)}, \dots, l_{u''(k)})$  を支配しているとき,  $L_u \leftarrow L_u \setminus \{(u'', l_{u''(1)}, \dots, l_{u''(k)})\}$  とする。

**Step 3.** 全てのパレート解を出力

## 第4章 解法の提案と実験的評価

この章では，多目的最短経路問題についての分析を行い解法の考察をする．その後，提案解法の紹介と実装における工夫，本研究に対する成果を述べる．

### 4.1 問題の分析

本研究では多目的単一始点最短経路問題を扱う．多目的単一始点最短経路問題とは特定の1つのノードから他の全ノードとの間の最短経路問題であり，入力と出力は以下のようになる．

入力：重み付きグラフ，始点  $s$ ，最適化目的の数  $k$

出力： $s$  から全頂点へのパレート最適解となる経路の集合

パレート最適解の集合の数に対する問題の難しさ

一般的に多目的最適化問題は出力となるパレート最適解の数に比例して解くことが難しくなる．これは経路の探索をするとともに，パレート最適解であるかどうかの判定のために既知の経路と比較をするためである．単目的最短経路問題の場合，1つの頂点に1つの経路しか存在しないため既知の経路との比較は1回である．しかし，多目的最短経路問題の場合，1つの頂点に対して始点と終点を結ぶ経路の本数分の経路が存在する．任意の頂点  $v \in V$  を選択する．頂点集合  $V$  から始点  $s$  と  $v$  を除いた頂点集合を  $V' = V \setminus \{s\}$  とする．経路は始点から終点への通過する頂点の順番で表されるため，通過する頂点の集合の並び替えの数だけ存在する． $s$  から  $v$  への経路の数は  $V'$  から  $i$  個の頂点を選択し並び変えた数だけ存在する．頂点数を  $|N| = n$  とすると  $V'$  の要素数は  $|V'| = n - 2$  なので， $\sum_{i=0}^{n-2} (n-2)P_i$  で表される．よって，既知の経路との比較は最大で  $\sum_{i=0}^{n-2} (n-2)P_i$  回である．最適化目的の数を  $k$  とすると既知の経路との比較は，単目的最短経路問題では1回，計算量は  $O(1)$  であり，多目的最短経路問題では最大で  $\sum_{i=0}^{n-2} (n-2)P_i$  回，計算量は  $O(k * n^1)$

であるためパレート最適解の数に比例して解くことが難しくなる． ( $O(\sum_{i=0}^n i) = O(n)$ ,  $O(nP_i) = O(n^i)$  より,  $O(\sum_{i=0}^{n-2} (n-2)P_i) = O(k * n^{n-1})$  ) パレート最適解の数に比例して解くことが難しくなることを照明したため, それぞれのインスタンスに対してパレート最適解の数がどのように変化するか分析する．

### 頂点数に対する問題の難しさ

パレート最適解の数が問題の難しさに直結することを示した．頂点数に対する問題の難しさを分析するために, 入力 of 頂点数  $|V| = n$  によりパレート最適解の数がどのように変化するか分析する．頂点数とパレート最適解の集合の数の関係进行分析するためにその他の入力 is 辺の重みベクトルの値がランダムであり, 重みの範囲が十分に大きい完全グラフとする．多目的最短経路問題の場合, 解は始点から各頂点への経路の本数分存在する．頂点集合  $V$  から始点  $s$  を除いた頂点集合を  $V' = V \setminus \{s\}$  とする．経路は始点から終点への通過する頂点の順番で表されるため, 通過する頂点の集合の並び替えの数だけ存在する．完全グラフにおいて始点から各頂点への経路の合計は  $V'$  から  $i$  個の頂点を選択し並び変えた数だけ存在する． $V'$  の要素数は  $|V'| = n - 1$  なので,  $\sum_{i=0}^{n-1} (n-1)P_i$  で表される．以上より, 頂点数が大きくなるとパレート最適解の数は指数的に増えるため, 頂点数が大きくなると問題は難しくなる．

### 重みの範囲に対する問題の難しさ

パレート最適解の数が問題の難しさに直結することを示した．重みの範囲に対する問題の難しさを分析するために, 全ての重みが非負である完全グラフにおいて重みの範囲によりパレート最適解の数がどのように変化するか分析する．重みの範囲によりパレート最適解の集合の数の関係进行分析するためにその他の入力 is 辺の重みベクトルの値がランダムであり, 頂点数が十分に大きい完全グラフとする．始点  $s$  から終点  $t$  への最短経路を求める．頂点  $v$  を経由した  $s$  から  $t$  への経路を  $(s, v, t)$  と表す．非負であるグラフにおいて, 経由する頂点数が多くなるほど各目的関数の値は大きくなるため, 経路  $(s, t)$  が経路  $(s, v, t)$  を支配する確率は高くなる．また, 経路  $(s, t)$  が  $i$  個の頂点を経由する経路  $(s, v_1, \dots, v_i, t)$  を支配する確率も高くなる．ここで, 重みの範囲が広くなると経路  $(s, v, t)$  が解となる確率は高くなるため解の数は多くなると予想される．経由する頂点 1 つの場合, 重みの範囲に対して解になる確率が高くなることを示したが, 経由する頂点数が増加

した場合でも同じく重みの範囲に対して解になる確率が高くなると予想されるため全体の解の数はさらに多くなると予想される。経路する頂点数が増えるほど解になる確率が低くなっていくが、重みの範囲が広がると解の確率は上がるため、一定の範囲で2つの確率が相殺しあうため、重みの範囲が一定の値を越えると解は増えなくなる。以上より、重みの範囲が広がると最適解の数は増えるため、重みの範囲が広がると問題は難しくなる。

証明：最適化目的の数が2 ( $f_1, f_2$ )，重みの範囲が1であるグラフを考える。任意の2頂点間の目的関数の組み合わせは  $f_1$  が 0, 1 の2通り，  $f_2$  が 0, 1 の2通りなので  $2 \times 2$  の4通りである。経路  $(s, t)$  が経路  $(s, v, t)$  の組み合わせは経路3本の組み合わせなので  $4^3 = 64$  通り。ここで、経路  $(s, t)$  が経路  $(s, v, t)$  を支配するまたは同値である組み合わせを考える。上記の組み合わせが成立するためには以下が成り立たなければならない。経路  $(s, t)$  の各目的関数  $f_{st1}, f_{st2}$ ，経路  $(s, v)$  の各目的関数  $f_{sv1}, f_{sv2}$ ，経路  $(v, t)$  の各目的関数  $f_{vt1}, f_{vt2}$  において  $f_{st1} > f_{sv1} + f_{vt1}$  または  $f_{st2} > f_{sv2} + f_{vt2}$ 。これは  $f_{st1} = 1$  かつ  $f_{sv1} + f_{vt1} = 0$ ，  $f_{st2} = 1$  かつ  $f_{sv2} + f_{vt2} = 0$  なので  $2^3 + 2^3 - 1 = 15$  通り。よって経路  $(s, v, t)$  が解となる確率は  $15/64 = 0.234$  である。最適化目的の数が2 ( $f_1, f_2$ )，重みの範囲が2であるグラフを考える。任意の2頂点間の目的関数の組み合わせは  $f_1$  が 0, 1, 2 の3通り，  $f_2$  が 0, 1, 2 の3通りなので  $3 \times 3$  の9通りである。経路  $(s, t)$  が経路  $(s, v, t)$  の組み合わせは経路3本の組み合わせなので  $9^3 = 729$  通り。ここで、経路  $(s, t)$  が経路  $(s, v, t)$  を支配するまたは同値である組み合わせを考える。上記の組み合わせが成立するためには以下が成り立たなければならない。経路  $(s, t)$  の各目的関数  $f_{st1}, f_{st2}$ ，経路  $(s, v)$  の各目的関数  $f_{sv1}, f_{sv2}$ ，経路  $(v, t)$  の各目的関数  $f_{vt1}, f_{vt2}$  において  $f_{st1} > f_{sv1} + f_{vt1}$  または  $f_{st2} > f_{sv2} + f_{vt2}$ 。これは  $(3^3 + 3^3 - 1) + (3 \times 3^3 + 3 \times 3^3 - 3) = 53 + 159 = 212$  通り。よって経路  $(s, v, t)$  が解となる確率は  $212/729 = 0.291$  である。以上より、重みの範囲が広がると経路  $(s, v, t)$  が解となる確率は高くなる。

### 最適化目的の数に対する問題の難しさ

パレート最適解の数が問題の難しさに直結することを示した。最適化目的の数に対する問題の難しさを分析するために、全ての重みが非負である完全グラフにおいて最適化目的の数によりパレート最適解の数がどのように変化するか分析する。最適化目的の数によりパレート最適解の集合の数の関係を分析するためにその他の入力値は辺の重みベクトルの値がランダムかつ重みの範囲が十分に大きい値であり、頂点数が十分に大きい完全グラフとする。始点  $s$  から終点  $t$  への最短経路を求める。頂点  $v$  を経由した  $s$  から  $t$  への経路を

$(s, v, t)$  と表す. 非負であるグラフにおいて, 経由する頂点数が多くなるほど各目的関数の値は大きくなるため, 経路  $(s, t)$  が経路  $(s, v, t)$  を支配する確率は高くなる. また, 経路  $(s, t)$  が  $i$  個の頂点を経由する経路  $(s, v_1, \dots, v_i, t)$  を支配する確率も高くなる. 最適化目的の数が 2 の場合と 3 の場合を比較する.  $(s, t)$  の各目的関数を  $f_{st1}, f_{st2}, f_{st3}$ ,  $(s, v, t)$  の各目的関数を  $f_{svt1}, f_{svt2}, f_{svt3}$  とする. 経路  $(s, v, t)$  が解となるためには 1 つでも目的関数が  $(s, t)$  より低ければ良い. つまり,  $\exists i, f_{svti} < f_{sti}$  が成り立てば良い. 最適化目的の数が 2 の場合  $i$  の選択肢は 2 つだが, 最適化目的の数が 3 の場合  $i$  の選択肢は 3 つとなり  $\exists i, f_{svti} < f_{sti}$  が成り立つ可能性が高くなる. よって, 最適化目的の数が増えると頂点を 1 つ経由した経路が解となる確率が高くなる. 経由する頂点数が増えた場合でも同じことが言えるので全体の解の数はさらに増えると予想される. 以上より, 最適化目的の数が増えると最適解の数は増えるため, 最適化目的の数が増えると問題は難しくなる.

#### 辺の本数に対する問題の難しさ

パレート最適解の数が問題の難しさに直結することを示した. 辺の本数に対する問題の難しさを分析するために, 全ての重みが非負であるグラフにおいて辺の本数によりパレート最適解の数がどのように変化するか分析する. 辺の本数によりパレート最適解の集合の数の関係を分析するためにその他の入力, 最適化目的の数が  $k$ , 辺の重みベクトルの値がランダム, 重みの範囲が十分に大きい, 頂点数が十分に大きいグラフとする. 始点  $s$  から終点  $t$  への最短経路を求める. 頂点  $v$  を経由した  $s$  から  $t$  への経路を  $(s, v, t)$  と表す. 完全グラフの場合, 頂点を 1 つ経由する経路は始点と終点を除いた頂点集合から 1 頂点を選択した経路なので  $n - 2$  通り存在する. グラフ上の  $s$  と  $t$  を除いた頂点集合の任意の頂点  $v'$  について,  $s$  と  $v'$  を結ぶ辺が存在しない又は  $v'$  と  $t$  を結ぶ辺が存在しないとき, 頂点  $v'$  を通る経路  $(s, v, t)$  は存在しない. また, 頂点  $u$  と頂点  $u'$  を結ぶ辺が存在しないとき,  $(s, \dots, u, u', \dots, t)$  となるような経路は存在しない. このように辺の本数が少なくなると, グラフ上の経路は少なくなるため解の候補となる経路の数が減少する. よって, 全体の解の数は少なくなる. 単一始点最短経路問題において, 始点  $s$  を中心としたスターグラフ内の解となり得る経路は  $n - 1$  本存在する. 単一始点最短経路問題において, 完全グラフ内の解となり得る経路は  $\sum_{i=0}^{n-1} (n-1)P_i$  本存在する. 以上より, 辺の本数が増えると最適解の数は増えるため, 辺の本数が増えると問題は難しくなる.

#### 各目的関数間の相関に対する問題の難しさ

パレート最適解の数が問題の難しさに直結することを示した．各目的関数間の相関に対する問題の難しさを分析するために，全ての重みが非負である完全グラフにおいて各目的関数間の相関によりパレート最適解の数がどのように変化するか分析する．各目的関数間の相関によりパレート最適解の集合の数の関係を分析するためにその他の入力，最適化目的の数が3，辺の重みベクトルの値がランダム，重みの範囲が十分に大きい，頂点数が十分に大きい完全グラフとする．相関は $-1 \sim 1$ で表され， $-1$ に近いほど負の相関があり， $0$ に近いほど相関がなく， $1$ に近いほど正の相関があるという．以下では頂点に入ってくる辺の重みベクトルに対する相関と頂点から出ていく辺の重みベクトルに対する相関に対する分析を行う．

頂点に入ってくる辺の重みベクトルに対する相関に対する分析．最適化目的の数3のため，各目的関数を $(f_1, f_2, f_3)$ とし，任意の辺 $e \in E$ における重みベクトルを $\vec{e} = (e(1), e(2), e(3))$ とする．終点が頂点 $u$ である辺（頂点 $u$ に入ってくる辺）の集合を $E_u$ とする．頂点に入ってくる辺の重みベクトルに対する相関とは3つある． $\forall E_v, v \in V$ に対する相関であり，それぞれ $\{f_1, f_2\}$ 間の相関 $c_{12}$ ， $\{f_2, f_3\}$ 間の相関 $c_{23}$ ， $\{f_1, f_3\}$ 間の相関 $c_{13}$ である． $c_{12} = 1$ のとき，任意の辺 $x, v \in E$ に対して， $x(1) = x(2) * l$ のとき $y(1) = y(2) * l$ が成り立つ．つまり， $c_{12} = 1$ のとき，任意の辺 $x, v \in E$ に対して， $x(1) < x(2) \wedge y(1) < y(2)$ または $x(1) = x(2) \wedge y(1) = y(2)$ または $x(1) > x(2) \wedge y(1) > y(2)$ が成り立つ． $c_{12} = 1, c_{23} = 1$ のとき $c_{13} = 1$ である．このとき，任意の辺 $e \in E$ を $\vec{e} = (e(1), e(2), e(3))$ とすると $\forall e' \in E, \vec{e}' = l * (e(1), e(2), e(3))$ が成り立つため各頂点に対する解の数はそれぞれ1となり，全体の解の数は $|V|$ となる．また，各目的関数間の相関が強ければ強いほど頂点 $u \in V$ における $s$ からの経路 $u'_s$ と $u''_s$ に対して，支配関係（ $u'_s$ が $u''_s$ を支配するまたは $u''_s$ が $u'_s$ を支配する）が成り立つ可能性が高いため全体の経路の数は少なくなる．以上より，頂点に入ってくる辺の重みベクトルに対する相関が弱いと最適解の数は増えるため，頂点に入ってくる辺の重みベクトルに対する相関が弱いと問題は難しくなる．

頂点から出ていく辺の重みベクトルに対する相関に対する分析．最適化目的の数3のため，各目的関数を $(f_1, f_2, f_3)$ とし，任意の辺 $e \in E$ における重みベクトルを $\vec{e} = (e(1), e(2), e(3))$ とする．始点が頂点 $u$ である辺（頂点 $u$ から出ていく辺）の集合を $E_u$ とする．頂点から出ていく辺の重みベクトルに対する相関とは3つある． $\forall E_v, v \in V$ に対する相関であり，それぞれ $\{f_1, f_2\}$ 間の相関 $c_{12}$ ， $\{f_2, f_3\}$ 間の相関 $c_{23}$ ， $\{f_1, f_3\}$ 間の相関 $c_{13}$ である． $c_{12} = 1$ のとき，任意の辺 $x, v \in E$ に対して， $x(1) = x(2) * l$ のと

き  $y(1) = y(2) * l$  が成り立つ．つまり，  $c_{12} = 1$  のとき，任意の辺  $x, v \in E$  に対して，  $x(1) < x(2) \wedge y(1) < y(2)$  または  $x(1) = x(2) \wedge y(1) = y(2)$  または  $x(1) > x(2) \wedge y(1) > y(2)$  が成り立つ．  $c_{12} = 1, c_{23} = 1$  のとき  $c_{13} = 1$  である．このとき，任意の辺  $e \in E$  を  $\vec{e} = (e(1), e(2), e(3))$  とすると  $\forall e' \in E, \vec{e'} = l * (e(1), e(2), e(3))$  が成り立つため各頂点に対する解の数はそれぞれ 1 となり，全体の解の数は  $|V|$  となる．また，各目的関数間の相関が強ければ強いほど頂点  $u \in V$  における  $s$  からの経路  $u'_s$  と  $u''_s$  に対して，支配関係（ $u'_s$  が  $u''_s$  を支配するまたは  $u''_s$  が  $u'_s$  を支配する）が成り立つ可能性が高いため全体の経路の数は少なくなる．以上より，頂点から出ていく辺の重みベクトルに対する相関が弱いと最適解の数は増えるため，頂点から出ていく辺の重みベクトルに対する相関が弱いと問題は難しくなる．

### 多目的最短経路問題に対する単目的最短経路問題アルゴリズムの実装

入力が，頂点集合  $V$ （ $|V| = n$ ），最適化目的の数が  $k$ ，辺の重みベクトルの値がランダム（相関がない），重みの範囲が十分に大きい，完全グラフとする．本研究では重みがランダム（異なる）単一起点最短経路を扱うため，単一目的最短経路問題での解法はダイクストラ法とベルマンフォード法である．（重みが決まった値でないため幅優先探索は扱えない．全点对最短経路でないためワーシャルフロイド法は効率的でない．）多目的最短経路問題に対するダイクストラ法とベルマンフォード法の適用を実装し，それぞれの問題点と活用方法を分析する．

ダイクストラ法の適用．ダイクストラ法はすでに探索済みのノードの中で重みが最小のノードを求め，更新対象として探索していくアルゴリズムである．全ての重みが非負の場合，探索したノードの中で重みが最小のものはその後の探索で更新されることはないので重みが決定する．多目的最短経路問題にダイクストラ法を適用するとすでに探索済みの経路の中で重みが最小の経路を求め，更新対象として探索していく．全ての重みが非負の場合，探索した経路の中で重みが最小のものはその後の探索で支配されることはないので解として決定する．ダイクストラ法の問題点は重みが最小である経路の探索であり，理由は 2 つある．1 つ目の理由は多目的最適化問題において経路は複数の目的関数を所持しているため最小である基準が明確でないことである．単目的最短経路問題では目的関数が 1 つのためその目的関数が最小の経路（頂点）を選択すれば良い．目的関数が複数である多目的最短経路問題では，各目的関数に対して評価値をもうけ優先順位をつける方法（例：目的関数が  $f_1, f_2, f_3$  のとき  $f_2 \rightarrow f_1 \rightarrow f_3$  という優先順位をつける）や 1 つの目的関数として計算する方法（例：目的関数が  $f_1, f_2, f_3$  のとき  $f_x = f_1 + f_2 * 2 + f_3 * 4$  という目的関数



を計算する) が考えられる. もう 1 つの問題は重みが最小である経路 (更新対象) の探索するために多くの比較が必要となることである. 単目的最短経路問題の場合選択肢となる経路 (頂点) は頂点数分なので最大で  $|V|$  つあるが, 多目的最短経路の場合選択肢となる経路は 1 頂点に対して複数存在するため最大で  $\sum_{i=0}^{n-1} (n-1) P_i$  つとなる. 探索のたびにこれらの経路の中から 1 つの経路を選択するには毎回それぞれの値を比較して対象となる経路を求めなければならないため膨大な計算時間がかかる. これらの問題に対して経路が発見されるたびに更新順のリストに保存していく方法があるが, リストへの保存に対しても多くの計算時間がかかるため問題は難しいままである.

ベルマンフォード法の適用. ベルマンフォード法は頂点数を  $|V|$  とした時, 全辺を緩めることを単に  $|V| - 1$  回繰り返すアルゴリズムである. 多目的最短経路問題にベルマンフォード法を適用すると, 単目的最短経路問題と同じように頂点数を  $|V|$  とした時, 全辺を緩めることを単に  $|V| - 1$  回繰り返す. ダイクストラ法と違い, 更新対象を探索する操作がないためダイクストラ法のボトルネックとなっていた目的関数が最小の経路 (頂点) を選択するための比較や計算をしなくて済む. しかし, 全辺を緩める操作のときにすでに更新した経路まで更新対象としてしまうため無駄が生じてしまう. 単目的最短経路問題の場合, 各頂点に最大で 1 つの解 (経路) が存在するため 1 回の全辺を緩める操作で更新対象となるのは最大で頂点数である  $|V|$  である. 一方, 多目的最短経路問題では各頂点に最大で始点から経路の本数分 (完全グラフの場合, 存在する始点からの経路の本数は  $\sum_{i=0}^{n-2} (n-2) P_i$  本) 存在する. また, 1 回の全辺を緩める操作で更新対象となるのは最大で  $\sum_{i=0}^{n-1} (n-1) P_i$  つとなる. そのため, 全辺を緩めることを単に  $|V| - 1$  回繰り返してしまうと膨大な計算時間がかかってしまう.

## 4.2 非負の問題に対する解法の提案

非負である多目的最短経路問題における解法を提案する.

入力: グラフ  $G = (V, E)$ , 始点  $s \in V$ , 最適化目的の数  $k$ , 各辺の重みを返す関数  $w : E \rightarrow \mathbb{R}^k$

出力:  $s$  から全ての頂点への最短経路となるパレート解の集合

記号

$k$  : 最適化目的の数

$v \in V, j = 1, \dots, k$  に対して

$l_{v(j)}$  : 始点からノード  $v$  に到達したときに生じる第  $j$  番目の目的関数における  
総コスト

$e \in E, j = 1, \dots, k$  に対して

$e_{(j)}$  : 辺  $e$  の第  $j$  番目の目的関数におけるコスト

#### 4.2.1 提案解法 1

多目的最短経路問題に対する単目的最短経路問題アルゴリズムの実装で述べたようにダイクストラ法とベルマンフォード法にはそれぞれボトルネックがあるため改良が必要である。ダイクストラ法のボトルネックである更新対象の探索はヒープを用いた方法や経路が発見されるたびに更新順のリストに保存していく方法があるが目的関数が増えていくとこれらの操作時間にも計算時間がかかってしまうため現実的でない。一方、ベルマンフォード法は改善によってボトルネックを解決できる。そのため、最適な解法を得るためにベルマンフォード法の改善をしていく。

まず、ベルマンフォード法のボトルネックである全辺を緩めることを単に  $|V| - 1$  回繰り返してしまうと膨大な計算時間がかかってしまうという問題と、1 回の全辺を緩める操作で更新対象となるのは最大で  $\sum_{i=0}^{n-1} (n-1)P_i$  つとなる問題を解決した解法を以下に記載する。

記号

$L$  : 解となる経路のラベル

$X$  : 更新対象とする頂点集合

アルゴリズム

**Step 1.**  $\forall v \in V, L_v \leftarrow \emptyset, L_s \leftarrow (s, 0, \dots, 0), X \leftarrow s$

**Step 2.**  $X = \{\emptyset\}$  となるまで以下の操作を行う。

**Step 2-1.**  $v \in X$  となる頂点  $v$  を選択する。

**Step 2-2.**  $X \leftarrow X \setminus \{v\}$

**Step 2-3.**  $e = e \in E \mid v, u \in e$  となる  $e$  に対して以下の操作を行う。

**Step 2-3-1.** 頂点  $v$  に対する全ての経路  $(v', l_{v'(1)}, \dots, l_{v'(k)}) \in L, v = v'$  に対して以下の操作を行う。

**Step 2-3-1-1.** 辺  $e$  の重みベクトルを  $\vec{e} = (e_{(1)}, \dots, e_{(k)})$  とし,  
 $(u', l_{u'(1)}, \dots, l_{u'(k)}) \leftarrow (u', l_{v(1)} + e_{(1)}, \dots, l_{v(k)} + e_{(k)})$  とする。

**Step 2-3-1-2.** 以下の条件を満たすとき,  $L \leftarrow L \cup \{(u', l_{u'(1)}, \dots, l_{u'(k)})\}$ ,  
 $X \leftarrow X \cup \{u\}$  とする。

- 任意の  $(u^*, l_{u^*(1)}, \dots, l_{u^*(k)}) \in L, u = u^*$  に  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  が支配されない。
- 任意の  $(u^*, l_{u^*(1)}, \dots, l_{u^*(k)}) \in L, u = u^*$  と  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  における全ての目的関数値が同値でない。

**Step 2-3-1-3.** 任意の  $(u'', l_{u''(1)}, \dots, l_{u''(k)}) \in L, u = u''$  に対して  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  が  $(u'', l_{u''(1)}, \dots, l_{u''(k)})$  を支配しているとき,  $L \leftarrow L \setminus \{(u'', l_{u''(1)}, \dots, l_{u''(k)})\}$  とする。

**Step 3.** 全てのパレート解を出力

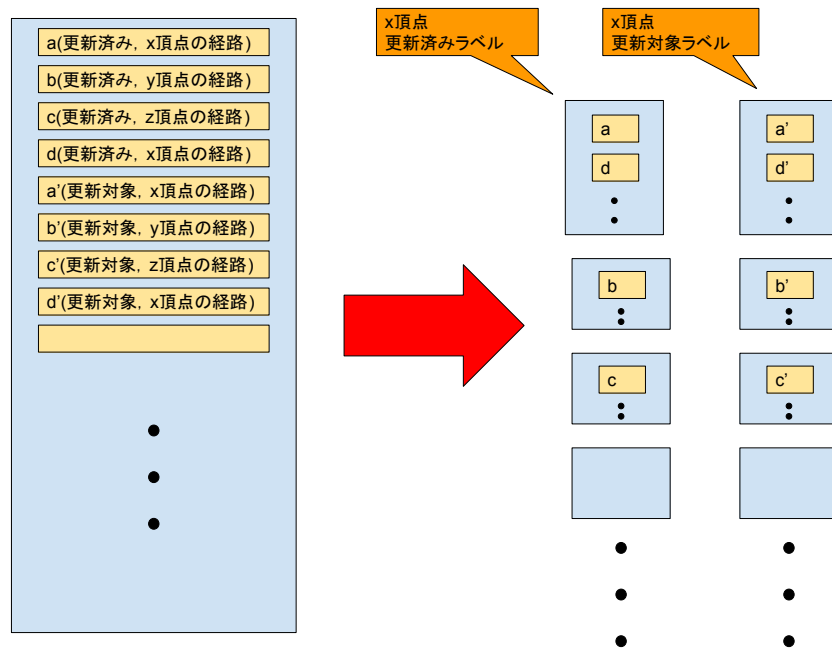
上記の解法は更新によって新たな解を得た頂点を更新対象となる頂点集合に加えることによって、更新しなくて良い頂点を更新せずに済む。これにより、1回の全辺を緩める操作で更新対象となるのは最大で  $\sum_{i=0}^{n-1} (n-1)P_i$  つとなる問題を解決した。これにより、全体の探索における経路の更新回数に対する合計計算量を  $O(n^{n+1})$  から  $O(n^n)$  に削減した。また、更新対象とする頂点が存在しなくなった時点で探索が終了するため全辺を緩めることを単に  $|V| - 1$  回繰り返してしまうと膨大な計算時間がかかってしまうという問題を解決した。しかし、この解法にはまだ改善点がいくつか存在するためそれらの改善をしていく。この解法の問題点は2つある。

1つ目はすでに更新済みである経路を更新対象にしているために無駄な更新が行われている点である。頂点に対する解である経路は複数存在している。頂点  $v$  に対してすでに更新済みである経路を  $a, b$  とし、他の頂点の更新により新しく得られた経路を  $c, d$  とすると、次に頂点  $v$  の更新を行うとき経路  $c, d$  のみを更新対象とすれば良い。上記の解法では頂点に対する経路  $a, b, c, d$  すべてが更新対象となってしまうため経路  $a, b$  の無駄な更新が行われてしまう。

2つ目はパレート最適解の判定の際に比較対象となる経路を探索するのに多くの計算時

間がかかってしまう点である．上記の解法では解となる経路を保存するラベルが1つのため比較対象となる経路を選択するためにすでに求めてある経路全てを探索する必要がある．全ての経路は最大で  $\sum_{i=0}^{n-1} (n-1)P_i$  存在し，比較対象とする経路は最大で  $\sum_{i=0}^{n-2} (n-2)P_i$  のため，選択のための探索に膨大な計算時間がかかってしまう．

図 4.1: ベルマンフォード改良解法と従来のラベル比較



上記の解法の問題点を解決するために，従来研究で提案された完全多項式時間近似スキームによる解法とラベル付けアルゴリズムによる解法を組み合わせ改良を加える．

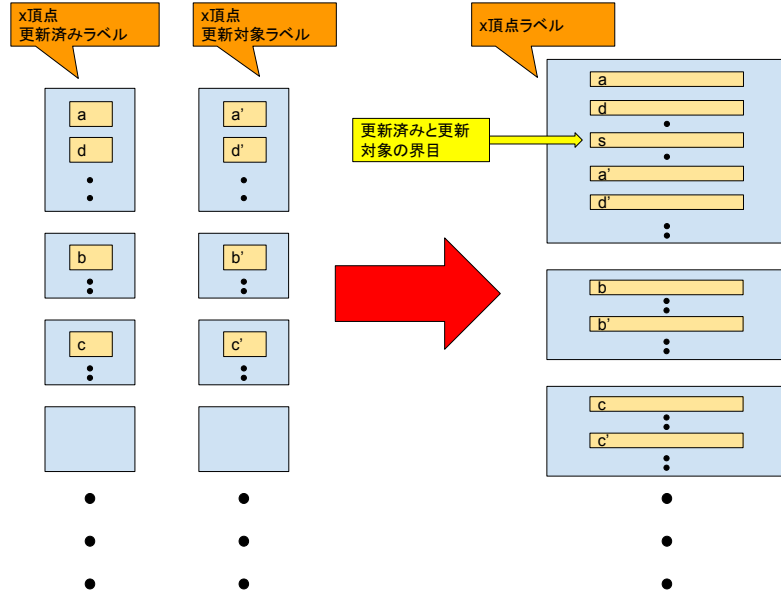
まず，完全多項式時間近似スキームによる解法で提案されたように，更新対象とする経路のラベルと更新対象としない経路のラベルを用意することによって無駄な更新を削減し効率化する．これにより経路  $c, d$  のみを更新対象とすることができるため経路  $a, b$  の無駄な更新を行わずにすむ．

つぎに，ラベル付けアルゴリズムによる解法で提案されたように頂点数だけラベルを用意し，各頂点毎の経路をそれぞれのラベルに記憶していく．従来のアルゴリズムでは1つのラベルから更新対象となる頂点に対する経路を探索し経路の更新を行っていたが，1つのラベルには全ての頂点に対する経路が存在するため多くの経路に対して更新対象とするかの判定を行わなければならない．

ラベル付けアルゴリズムによる解法により各頂点毎の経路を分けて保存しておくことが可能なので更新対象となる頂点に対する経路を選択するときに探索を行う必要がなくなり効率化になると考えられる．これにより，パレート解であるかの判定に要する計算量を

$O(k * n^n)$  から  $O(k * n^{n-1})$  に削減した.

図 4.2: ラベルの統合 (提案解法 1 のラベル)



完全多項式時間近似スキームによる解法とラベル付けアルゴリズムによる解法を組み合わせることにより無駄な更新を削減と経路を選択するための探索の削減を実現した. また, 全体の探索における更新に対する計算量を  $O(k * n^{2n+1})$  から  $O(k * n^{2n-1})$  に削減した. これは図に示すようにいくつかのラベルに経路を分配することにより可能にしている. しかし, ラベルを複数作ってしまうとメモリを多く使ってしまう.

この問題を解決するために任意の頂点に対する更新済みのラベルと更新対象のラベルの結合を考える. 具体的には, 更新済みラベルの後ろに更新対象とするラベルを結合し, 更新対象とするラベルの先頭を記憶していく. 通常, 更新対象となる経路は後に更新済みの経路となり, 新たな更新対象となる経路が発見される. つまり, 更新対象となる経路をラベルの後ろに加えていくことで更新済みの経路の前に更新対象となる経路が来ることはない. このため, 更新対象となる先頭の経路を記憶しておけば, その後に続く経路は全て更新対象となる経路である. 図 4. 2 に示すように経路  $s$  を更新対象となる先頭の経路とすると, その後にある経路  $a', d'$  は更新対象となる経路である. 以下にアルゴリズムを記載する.

記号

$v \in V$  に対して

$L_v$ : 頂点  $v$  に対するラベル

$M(L_v)$  : 頂点  $v$  に対するラベルにおいて, 更新対象とその他の境界となる経路 (  $M(L_v)$  が  $L_v$  の  $i$  番目の解のとき,  $i+1$  番目以降の経路は更新対象とする )

## アルゴリズム

**Step 1.**  $\forall v \in V, L_v \leftarrow \emptyset, L_s \leftarrow (s, 0, \dots, 0)$

**Step 2.** 更新ができなくなるまで以下の操作を行う.

**Step 2-1.**  $\forall v \in V$  となる頂点  $v$  に対して以下の操作を行う.

**Step 2-1-1.**  $\forall u \in V$  となる頂点  $u$  に対して以下の操作を行う.

**Step 2-1-1-1.**  $L_v$  から  $L_u$  に対しての更新を行う.

**Step 2-1-2.**  $M(L_v)$  を  $L_v$  の最後の解とする.

**Step 3.** 全てのパレート解を出力

### $L_v$ から $L_u$ に対しての更新

**Step 1.**  $L_v, L_u$ , 頂点  $v$  から頂点  $u$  への辺  $e$  を受け取る.

**Step 2.**  $M(L_v)$  より後にある全ての経路  $(v', l_{v'(1)}, \dots, l_{v'(k)}) \in L_v$  について以下の操作を行う.

**Step 2-1.** 辺  $e$  の重みベクトルを  $\vec{e} = (e_{(1)}, \dots, e_{(k)})$  とし,  $(u', l_{u'(1)}, \dots, l_{u'(k)}) \leftarrow (u', l_{v'(1)} + e_{(1)}, \dots, l_{v'(k)} + e_{(k)})$  とする.

**Step 2-2.** 以下の条件を満たすとき  $L_u \leftarrow L_u \cup \{(u', l_{u'(1)}, \dots, l_{u'(k)})\}$  とする.

- 任意の  $(u^*, l_{u^*(1)}, \dots, l_{u^*(k)}) \in L_u$  に  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  が支配されない.
- 任意の  $(u^*, l_{u^*(1)}, \dots, l_{u^*(k)}) \in L_u$  と  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  における全ての目的関数値が同値でない.

**Step 2-3.** 任意の  $(u'', l_{u''(1)}, \dots, l_{u''(k)}) \in L_u$  に対して  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  が  $(u'', l_{u''(1)}, \dots, l_{u''(k)})$  を支配しているとき  $L_u \leftarrow L_u \setminus \{(u'', l_{u''(1)}, \dots, l_{u''(k)})\}$  とする.  $(u'', l_{u''(1)}, \dots, l_{u''(k)})$  の前の経路が  $M(L_u)$  の場合,  $(u'', l_{u''(1)}, \dots, l_{u''(k)})$  の次の経路を  $M(L_u)$  の界とする.

## 4.2.2 提案解法 2

問題の分析で示した通り，多目的最短経路問題はインスタンスのグラフについて辺の本数が多いほどパレート解の数は多くなり，問題は難しくなる．つまり，インスタンスのグラフが完全グラフのとき問題は最も難しくなる．

一般的に最短経路問題では，インスタンスの重みが非負であるグラフについて，通る頂点数が少ない経路が解になる確率が高い．また，通る頂点数が多い経路が解になる確率は低い（一時解となっても後に他の経路に支配され削除される可能性が高い）．

最適化目的の数が2，重みの幅が  $w$  の場合の始点  $s$  から終点  $t$  への最短経路を求める．頂点  $v$  を経由した  $s$  から  $t$  への経路（経由頂点数が1の経路）を  $(s, v, t)$  と表す．頂点  $v_1, \dots, v_{l-1}$  を経由した  $s$  から  $t$  への経路（経由頂点数が  $l$  の経路）を  $(s, v_1, \dots, v_{l-1}, t)$  と表す． $(s, v, t)$  の各目的関数の値の幅は  $[0, 2w]$  であり， $(s, v_1, \dots, v_{l-1}, t)$  の各目的関数の値の幅は  $[0, lw]$  となる． $[0, lw]$  の値が  $[0, 2w]$  の値となる確率は  $2/l$  であるため  $[0, 2w] \subset [0, lw]$  となる確率が高いと言える．よって， $(s, v, t)$  の各目的関数が  $(s, v_1, \dots, v_{l-1}, t)$  の各目的関数より低くなる確率が高いので  $(s, v_1, \dots, v_{l-1}, t)$  が解になる確率が低く， $(s, v, t)$  が解になる確率が高い．解とならない経路を先に探索してしまうと，解となっても後に削除される経路を更新対象としてしまい，更新する無駄作業と削除する無駄作業が生じてしまう．以上より，経由する頂点数が少ない経路から順に探索した方が効率的だと考えられる．

図 4.3: 提案解法 1 による探索

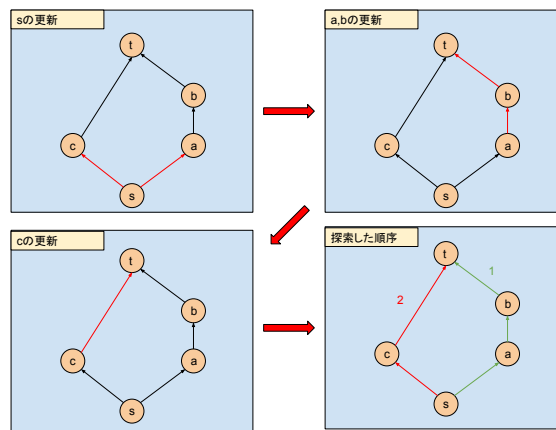
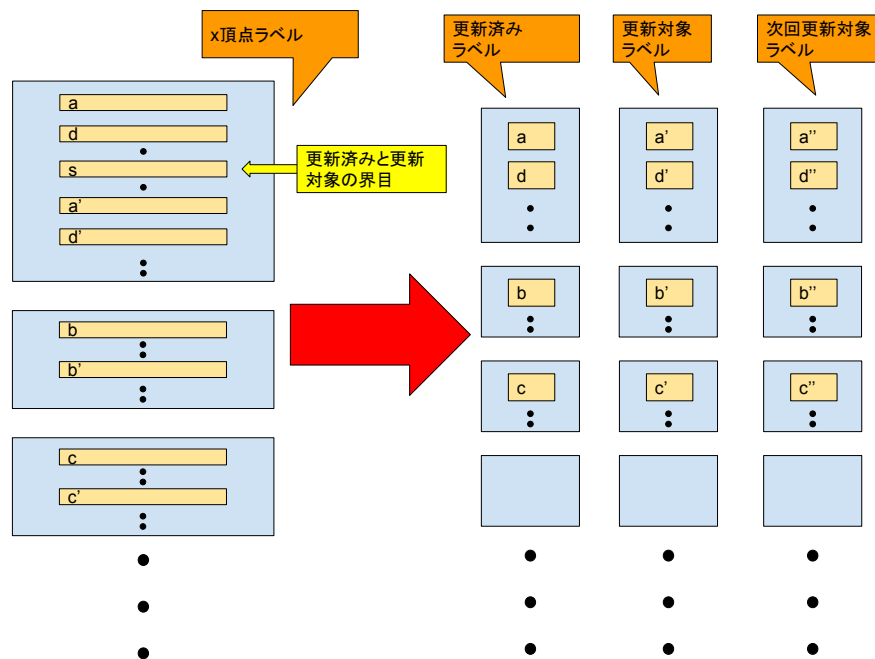


図 4.3 に示したように提案解法 1 の探索順序では経由する頂点が多い経路が経由する頂点が少ない経路よりも先に見つかってしまい 1 度ラベルに含まれてしまう．経由する頂点が多い経路は経由する頂点が少ない経路よりも解になる確率は低いため，経由する頂

点が少ない経路があとで探索されたとき経由する頂点が多い経路は削除される可能性がある。また、後に削除される経路に対して更新して求められた経路はパレート最適解とらない。後に削除される経路を  $a$ ,  $a$  を支配する経路を  $b$ ,  $a$  の更新によって求まる経路を  $a'$ ,  $b$  の更新によって求まる経路を  $b'$  とする。更新に使った辺を  $e$  とすると,  $a', b'$  はそれぞれ  $a, b$  の各目的関数に  $e$  の重みベクトルの値と足したものとなる。つまり,  $a' = a + e, b' = b + e$  が成り立つ。よって,  $a$  が  $b$  に支配されるとき  $a'$  は  $b'$  に支配されるため, 後に削除される経路に対して更新して求められた経路はパレート最適解とならない。経路に対して1度ラベルに含んだあと削除することや, 後に削除される経路に対して更新を行うという作業は無駄なため提案解法1の探索順序は非効率であると考えられる。

図 4.4: 提案1と提案解法2のラベル比較



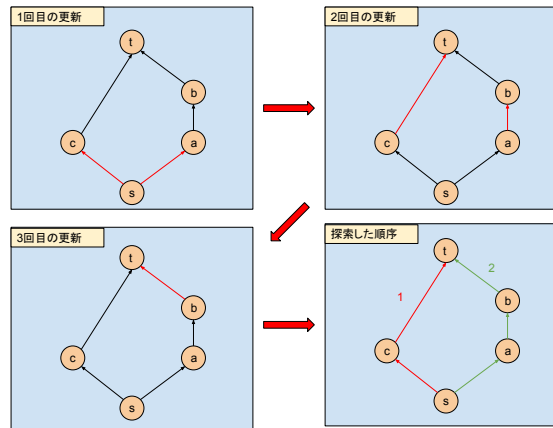
非効率である提案解法1の探索順序の改善方法を考える。すでに示した通り提案解法1の探索順序では経由する頂点が多い経路が経由する頂点が少ない経路よりも先に見つかってしまい1度ラベルに含まれてしまう。つまり, 経由する頂点が少ない経路が経由する頂点が多い経路よりも先に見つかるようにしたい。

まず, 提案解法1でそのようになる原因を考える。提案解法1の探索順序では1回の更新で全ての頂点に存在する更新対象となる経路を更新するが, 更新する頂点の順序によって経由する頂点が多い経路が更新されてしまう。例えば完全グラフにおいて頂点  $a, b$  を更新するとき,  $a$  を先に更新したとすると  $b$  には  $a$  に対する経路を更新して得られた経路が更新対象となる経路として保存されているため  $b$  の更新の際に  $a$  を経由して得られ



た経路（経由する頂点数が多い経路）を更新してしまう．この問題を解決するために 1 回の更新内で得られた経路（経由する頂点数が多い経路）を別で保存し，全頂点に対する更新が終わった時点で更新対象となる経路とする．これにより，全体の中で最長の経路（経由する頂点数が多い経路）は今までに行なった前回に対する更新の回数となり，経由する頂点が多い経路が経由する頂点が少ない経路よりも先に発見されることはない．

図 4.5: 提案解法 2 による探索



この探索方法により経由する頂点数が多い経路が経由する頂点が少ない経路よりも先に発見されることはなくなり，1 度解となった経路が後に削除される確率は低くなったので全体の経路に対する更新は少なくなると考えられる．しかし，1 回の更新内で得られた経路（経由する頂点数が多い経路）を別で保存し，全頂点に対する更新が終わった時点で更新対象となる経路とすることにより 1 回の更新に要する実装時間は長くなる．

提案解法 1 では更新対象となる経路を更新対象としない経路に移動させるだけだったが，提案解法 2 ではそれらに加えて更新内で得られた経路を更新対象とする経路に移動させるという作業が増える．そのため，提案解法 2 が提案解法 1 に比べて一定以上全体の経路に対する更新は少なくならなければ非効率となってしまふ．つまり，提案解法 2 が提案解法 1 に比べて一定以上削除される解を減らさなければならない．以下にアルゴリズムを記載する．

## 記号

$v \in V$  に対して

$L_{vx}$  : 頂点  $v$  に対する，更新対象としないラベル

$L_{vy}$  : 頂点  $v$  に対する，更新対象とするラベル

$L_{vz}$  : 頂点  $v$  に対する, 次回探索で更新対象とするラベル

## アルゴリズム

**Step 1.**  $\forall v \in V, L_v \leftarrow \emptyset, L_{sy} \leftarrow (s, 0, \dots, 0)$

**Step 2.** 更新ができなくなるまで以下の操作を行う.

**Step 2-1.**  $\forall v \in V$  となる頂点  $v$  に対して以下の操作を行う.

**Step 2-1-1.**  $\forall u \in V$  となる頂点  $u$  に対して以下の操作を行う.

**Step 2-1-1-1.**  $L_{vy}$  から  $L_u$  に対しての更新を行う.

**Step 2-2.**  $\forall v \in V$  に対して  $L_{vx}, L_{vy}, L_{vz}$  の更新を行う.

**Step 3.** 全てのパレート解を出力

$L_{vy}$  から  $L_u$  に対しての更新

**Step 1.**  $L_{vy}, L_u$ , 頂点  $v$  から頂点  $u$  への辺  $e$  を受け取る.

**Step 2.**  $\forall (v', l_{v'(1)}, \dots, l_{v'(k)}), (v', l_{v'(1)}, \dots, l_{v'(k)}) \in L_{vy}$  について以下の操作を行う.

**Step 2-1.** 辺  $e$  の重みベクトルを  $\vec{e} = (e_{(1)}, \dots, e_{(k)})$  とし,  $(u', l_{u'(1)}, \dots, l_{u'(k)}) \leftarrow (u', l_{v'(1)} + e_{(1)}, \dots, l_{v'(k)} + e_{(k)})$  とする.

**Step 2-2.** 以下の条件を満たすとき,  $L_{uz} \leftarrow L_{uz} \cup \{(u', l_{u'(1)}, \dots, l_{u'(k)})\}$  とする.

- 任意の  $(u^*, l_{u^*(1)}, \dots, l_{u^*(k)}) \in \{L_{ux} \cup L_{uy} \cup L_{uz}\}$  に  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  が支配されない.
- 任意の  $(u^*, l_{u^*(1)}, \dots, l_{u^*(k)}) \in \{L_{ux} \cup L_{uy} \cup L_{uz}\}$  と  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  における全ての目的関数値が同値でない.

**Step 2-3.** 任意の  $(u'', l_{u''(1)}, \dots, l_{u''(k)}) \in L_{ux}$  に対して  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  が  $(u'', l_{u''(1)}, \dots, l_{u''(k)})$  を支配しているとき  $L_{ux} \leftarrow L_{ux} \setminus \{(u'', l_{u''(1)}, \dots, l_{u''(k)})\}$  とする.

**Step 2-4.** 任意の  $(u'', l_{u''(1)}, \dots, l_{u''(k)}) \in L_{uy}$  に対して  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  が  $(u'', l_{u''(1)}, \dots, l_{u''(k)})$  を支配しているとき  $L_{uy} \leftarrow L_{uy} \setminus \{(u'', l_{u''(1)}, \dots, l_{u''(k)})\}$  とする.

**Step 2-5.** 任意の  $(u'', l_{u''(1)}, \dots, l_{u''(k)}) \in L_{uz}$  に対して  $(u', l_{u'(1)}, \dots, l_{u'(k)})$  が  $(u'', l_{u''(1)}, \dots, l_{u''(k)})$  を支配しているとき  $L_{uz} \leftarrow L_{uz} \setminus \{(u'', l_{u''(1)}, \dots, l_{u''(k)})\}$  とする.

$L_{vx}, L_{vy}, L_{vz}$  の更新

**Step 1.**  $L_{vx}, L_{vy}, L_{vz}$  を受け取る.

**Step 2.**  $L_{vx} = L_{vx} \cup L_{vy}$  とし,  $L_{vy} = \{\emptyset\}$  とする.

**Step 3.**  $L_{vy} \leftarrow L_{vz}$  とし,  $L_{vz} = \{\emptyset\}$  とする.

### 4.3 負の重みを考慮した問題に対する解法の提案

単目的最短経路問題では負の閉路が存在する場合、負の閉路の存在を報告し最短経路は求めなかった。これは負の閉路を何度も通過することによって重みを更新し続けるためである。しかし、多目的最短経路問題の場合、目的関数が複数存在するので1つの目的関数において負の閉路が存在する場合でも、他の目的関数による最適化をすることで解を求めることができる。よって、本研究では負の閉路が存在する目的関数を考慮しない解を求める。

単目的最短経路問題におけるベルマンフォード法では、 $|V|$  回目の更新を行うことによって負の閉路が存在するかの判定をし報告を行っていた。しかし、本研究では負の閉路が存在するとき負の閉路が存在しない目的関数に対しての解を求めるため、どの目的関数において負の閉路が存在しているのかを確かめる必要がある。また、負の閉路が存在するとき単目的最短経路問題よりも多くの実装時間を要してしまう。単目的最短経路問題では全体の解の数が最大で頂点数  $|V|$  のため1回の探索で最大で  $|V|$  つの経路の更新で済むが、多目的最短経路問題では全体の解の数が最大で  $\sum_{i=0}^{n-1} (n-1)P_i$  存在するため1回の探索で更新する経路は多くなるため  $|V|$  回目の更新を行うまでに多くの実装時間がかかってしまう。これらの問題を解決するために負の重みを考慮した問題に対する解法の提案を行う。また、提案解法2に基づいた解法を提案する。ある目的関数において負の閉路が存在する場合、その目的関数を削除する。目的関数の削除は以下の手順で行こなわれる。

探索途中で負の閉路を検出

単目的最短経路問題では全体の解の数が最大で頂点数  $|V|$  のため1回の探索で最大で  $|V|$  つの経路の更新で済むが、多目的最短経路問題では全体の解の数が最大で  $\sum_{i=0}^{n-1} (n-1)P_i$

つ存在するため 1 回の探索で更新する経路は多くなるため  $|V|$  回目の更新を行うまでに多くの実装時間がかかってしまうという問題に対して、探索途中で負の閉路を検出する方法を提案する。探索途中で負の閉路を検出することにより  $|V|$  回目の更新を行う必要がなくなり、 $|V|$  回目の更新を行うまでにかかる多くの実装時間が削減される。探索途中で負の閉路を検出する方法としては、提案解法 2 における Step 2-1-1-1 で負の閉路が存在するかの判定をし、1 回の全体に対する更新が終わった時点で負の閉路が存在する場合、その目的関数を削除する。

負の閉路が存在するかの判定は解となった経路が閉路となっているかで判定する。経路が閉路となっている場合、同じ頂点を 2 回通っているということである。同じ頂点を 2 回通っている経路が解となると、その頂点を 1 回通った経路に 2 回通った経路が支配されていない状態なので、任意の目的関数において 2 回通った経路の値が 1 回通った経路の値よりも低くなっている。2 回通った経路の値が 1 回通った経路の値よりも低くなっているということは 3 回通った経路も 2 回通った経路よりも低い値となり、負の閉路となる。よって、解となった経路が閉路となっているかで負の閉路の存在を判定できる。また、どの目的関数が低くなっているかを求めることで負の閉路の原因となっている目的関数を特定できる。

#### 目的関数 $k_1$ の削除

**Step 1.**  $k_1, E, L$ （全てのラベル）を受け取る。

**Step 2.**  $\forall e \in E$  について  $\vec{e} = f_1, \dots, f_k$  とし、 $k_1$  に対応する値が  $f_1$  のとき、 $f_1 = 0$  とする。

**Step 3.**  $\forall l \in L$  について  $\vec{l} = f_1, \dots, f_k$  とし、 $k_1$  に対応する値が  $f_1$  のとき、 $f_1 = 0$  とする。

**Step 4.**  $\forall l \in L$  について  $l$  が解であるかの判定をする。

#### $l$ が解であるかの判定

以下の条件を満たすとき  $L = L \setminus \{l\}$

- 任意の  $l' \in L$  に  $l$  が支配される。
- 任意の  $l' \in L$  と  $l$  における全ての目的関数値が同値である。

負の閉路探索を行なった後探索をする

どの目的関数において負の閉路が存在しているのかを確かめる必要があり、単目的最短経路問題では全体の解の数が最大で頂点数  $|V|$  のため 1 回の探索で最大で  $|V|$  つの経路の更新で済むが、多目的最短経路問題では全体の解の数が最大で  $\sum_{i=0}^{n-1} (n-1)P_i$  存在するため 1 回の探索で更新する経路は多くなるため  $|V|$  回目の更新を行うまでに多くの実装時間がかかってしまうという 2 つの問題を解決するために負の閉路探索を行なった後探索をする方法を提案する。探索前に負の閉路を検出することにより  $|V|$  回目の更新を行う必要がなくなり、 $|V|$  回目の更新を行うまでにかかる多くの実装時間が削減される。探索前に負の閉路を検出する方法としては、提案解法 2 における Step 1 の前で負の閉路が存在するかの判定をし、負の閉路が存在する場合その目的関数を削除する。負の閉路が存在するかの判定は各目的関数に対して単目的最短経路問題のベルマンフォード法を実行する。単目的最短経路問題のベルマンフォード法は  $|V|$  回の更新を行うが、1 回の更新で最大でも  $|V|$  つの更新しか行わないため短い時間で負の閉路の探索ができる。また、各目的関数ごとに負の閉路判定ができるため、どの目的関数において負の閉路が存在するのか分かる。

#### 目的関数 $k_1$ の削除

**Step 1.**  $k_1, E, L$  (全てのラベル) を受け取る。

**Step 2.**  $\forall e \in E$  について  $\vec{e} = e_{(1)}, \dots, e_{(k)}$  とし、 $k_1$  に対応する値が  $e_{(1)}$  のとき、 $e_{(1)} = 0$  とする。

#### 負の閉路探索結果で求めた解を利用して探索をする

前に提案した負の閉路探索を行なった後探索をする方法を改良する。負の閉路探索を行なった後探索をする方法では探索した経路は保存せずに最初から探索を行っていた。しかし、負の閉路で探索した解は少しの工夫で必ずパレート最適解に含まれるため、負の閉路探索結果で求めた解を利用して探索をする方法を提案する。

最終的にパレート最適解でない経路を利用して探索してしまうと、求めた解を利用して探索して得られた経路が削除されてしまうため無駄な探索となってしまう。よって、負の閉路探索結果で求めた解を利用して探索をする場合確実にパレート最適解となる解を求めて探索をしなければならない。

各目的関数に対して負の閉路探索で得た解はその目的関数の値に対して低くなる経路は存在しない。しかし、その目的関数の値が同じで他の目的関数が得た解より低い場合、得た解は支配されてしまいパレート最適解ではなくなってしまう。そのため、同じ値であ

る経路が得られた場合は他の目的関数の比較をし、優位な方を選択する必要がある。これにより、得られた解は必ずパレート解になるため求めた解を利用して探索をすることが可能になる。この解法は提案解法 2 に以下を付け加える。

**Step 0.** 各目的関数に対して以下の操作を行う。

**Step 0-1.**  $s_w = 0$  とし、 $v \in V/\{s\}$  に対して  $v_w = \infty$  とする。

**Step 0-2.**  $|V| - 1$  回以下の操作を行う。

**Step 0-2-1.**  $e = e \in E \mid v, u \in e$  となる  $e$  に対して、 $u_w > v_w + e_w$  を満たすとき以下の操作を行う。

**Step 0-2-1-1.**  $u_w = v_w + e_w$  .

**Step 0-2-2.**  $e = e \in E \mid v, u \in e$  となる  $e$  に対して、 $u_w = v_w + e_w$  を満たすとき以下の操作を行う。

**Step 0-2-2-1.**  $u_w$  となる経路が  $v_w + e_w$  となる経路に支配されるとき、 $u_w = v_w + e_w$  .

**Step 0-3.** Step 0-2-1 を行いノードの重みが更新された場合、その目的関数を削除する。

目的関数  $k_1$  の削除

**Step 1.**  $k_1, E, L$  (全てのラベル) を受け取る。

**Step 2.**  $\forall e \in E$  について  $\vec{e} = f_1, \dots, f_k$  とし、 $k_1$  に対応する値が  $f_1$  のとき、 $f_1 = 0$  とする。

**Step 3.**  $\forall l \in L$  について  $\vec{l} = f_1, \dots, f_k$  とし、 $k_1$  に対応する値が  $f_1$  のとき、 $f_1 = 0$  とする。

**Step 4.**  $\forall l \in L$  について  $l$  が解であるかの判定をする。

## 4.4 実装における工夫

リストの工夫

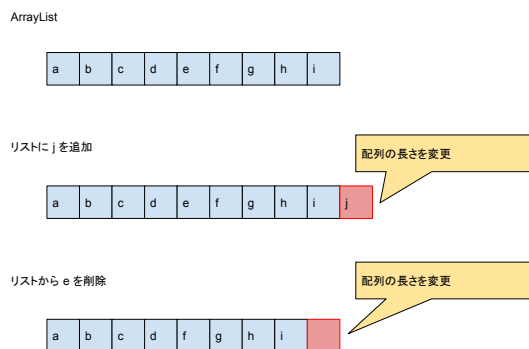
単目的最短経路問題では各頂点が 1 つの解（経路）しか持たないが、多目的最短経路問題では各頂点がパレート最適解として複数の解を持つ。そこで、複数の解を保存しておく

リストを作成する．リストは経路の集合である．多目的最短経路問題では1度解となった経路が後に発見される経路によって支配され削除されることがある．よって，リストを作成する際，効率的に解を削除することを考慮する．また，提案解法1では順序付きの集合として定義する必要がある．順序付きの集合として定義することによって更新する解をリストの前から選択していくことでどの解を選択していくかという作業がなくなるため提案解法2でも有効的に使用できる．

## ArrayList

ArrayList は配列を利用したリスト構造である．ArrayList は通常の配列と異なりサイズを後からでも変更できる．インデックス値による値の読み書きは高速である．しかし，要素の挿入/削除は，配列サイズが大きくなるほど，また，操作位置が先頭に近くなるほど遅くなる．

図 4.6: ArrayList のイメージ



提案解法1に対して，1つの頂点に対して1つの ArrayList を使用する．1つの ArrayList は1つの頂点に対する経路を保存しておき，更新対象としない経路と更新対象とする経路の界を記憶しておくことで更新対象のする経路の区別ができる．提案解法2に対して，1つの頂点に3つの ArrayList を使用する．

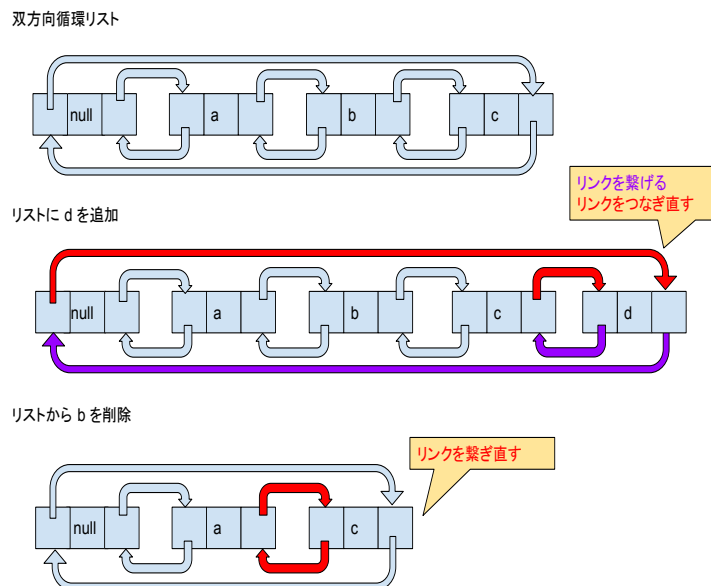
1つの頂点に対する各 ArrayList は更新対象としないもの，更新対象とするもの，次回の更新で更新経路対象にするものとする．提案解法1では更新対象としない経路と更新対象とする経路の界を移動させるだけで更新対象とするものから更新対象としないものへの経路の移動が可能であるが，提案解法1では更新対象とする経路を1つずつ更新対象としない経路に移動させるため時間がかかる．また，次回更新対象とする経路を更新対象とする経路に移動させるにも時間がかかる．経路の追加/削除については配列の長さを変更

させて行うため多少の時間がかかり、削除においては削除された解の分を詰めた配列にするためさらに時間がかかる。経路の数が変わるたびに配列の長さを変えられるため事前に解の数を予測する必要がない。

## class を用いた双方向循環リスト

双方向リストは連結リストである。各ノードは前のノードを示す後方リンクと後ろのノードを示す前方リンクを持つ。リストの先頭のノードには前のノードがないため、後方リンクには null 値を格納するか空のリストを指す。リストの最後尾のノードには次のノードがないため、前方リンクには null 値を格納するか空のリストを指す。循環リストは先頭と最後尾のノードを相互に連結した連結リストである。双方向循環リストは双方向リストと循環リストを組み合わせたものであり、全てのノードが前のノードを示す後方リンクと後ろのノードを示す前方リンクを持つ。双方向循環リストを用いることにより、経路の追加/削除にかかる作業と解の移動にかかる作業が容易になる。

図 4.7: 双方向循環リストにおける追加/削除のイメージ

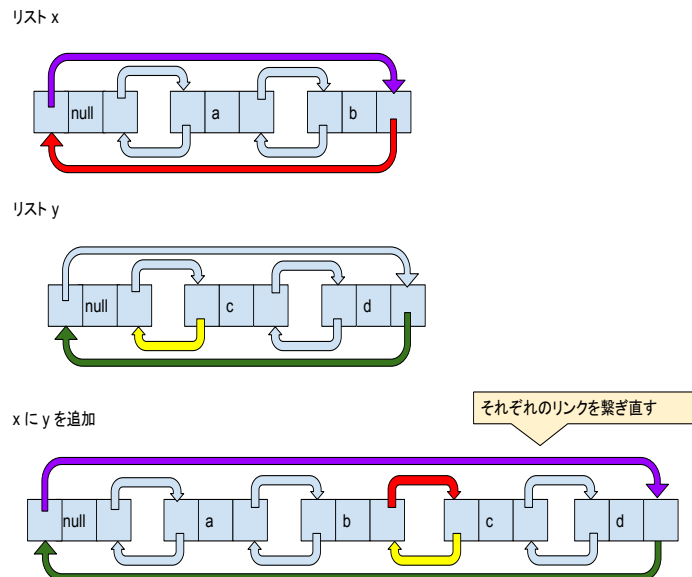


双方向循環リストを用いることにより、経路の追加/削除にかかる作業が容易になる理由を説明する。リストの先頭となる空のノード（先頭の前のノードかつ最後尾のノードの後ろのノード）を  $s$  とする。双方向循環リストで経路をリストの最後尾に追加するとき、リストの最後尾のノードを求めるのではなく、 $s$  の前のノードとして追加することができる。  $s$  は最後尾のノードの後ろのノードのため、  $s$  の前のノードはリストの最後尾のノード



ドとなる．よって， $s$  の前のノードとして追加することで，リストの最後尾のノードの後ろに追加することとなる．双方向循環リストでは，リストから経路を削除するとき配列のように長さを変更したりノードをつめるという作業が必要なくなる．リストから経路を削除するときの作業は削除するノードの前のノードの前方リンクを削除するノードの後ろのノードとし，削除するノードの後ろのノードの後方リンクを削除するノードの前のノードとする．この操作により，リンク内に含まれる他のノードを移動させたり，リンク自体に変更を加えることがない．以上より，双方向循環リストを用いることにより，経路の追加/削除にかかる作業が容易になる．

図 4.8: 双方向循環リストにおけるリスト間の追加イメージ



双方向循環リストを用いることにより，経路の移動にかかる作業が容易になる理由を説明する．リスト  $a$  の要素をリスト  $b$  に追加することを考える． $a$  の先頭となる空のノード（先頭の前のノードかつ最後尾のノードの後ろのノード）を  $a_s$  とし， $b$  の先頭となる空のノードを  $b_s$  とする．双方向循環リストにおいて， $a$  に存在するノード全てを  $b$  のリストの最後尾に追加するとき以下の作業を行う．

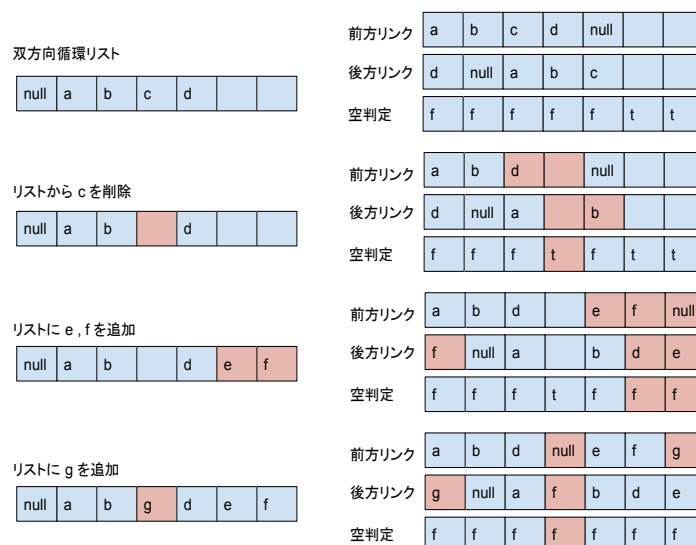
- $a_s$  の後ろノードの後方リンクを  $b_s$  の前のノードとし， $b_s$  の前のノードの後方リンクを  $a_s$  の前のノードとする．
- $a_s$  の前のノードの前方リンクを  $b_s$  とし， $b_s$  の後方リンクを  $a_s$  の前のノードとする．

双方向循環リストでは、ノードを1つずつ移動させるのではなく4つのリンクを変更することによって全てのノードを移動させることができる。この操作により、リンク内に含まれる他のノードを移動させたり、リンク自体に変更を加えることがない。classを用いた双方向循環リストの場合、解（経路）が発見されるたびに新しいノードを作成するため事前に解の数を予測する必要がない。

## 配列を用いた双方向循環リスト

配列を用いた双方向循環リストでは class を用いた双方向循環リスト同様双方向循環リストを使用する。配列を用いた双方向循環リストは class が使えないプログラミング言語（C 言語等）でも使用できる。配列を用いているため、事前に解の数を予測する必要がある。解の数が用意した配列を超える場合、配列の長さが長い配列を作成し要素となっている経路を全て移動させる必要があるため非効率となる。配列で経路を削除するときリンクを書き換えることにより可能になるが、削除される解は配列内に残ってしまう。削除される経路が配列内に残っているとメモリを大量に使用してしまうため、できるだけ削除される経路が残されない効率的な配列としたい。そのため、削除された経路が存在している配列の場所を記憶しておき、経路が発見されたときにその場所に格納する。経路が発見されたときに毎回削除された解の場所を探し格納するのは非効率なため、配列が全て埋まった時点で削除された経路の場所を探し格納する操作を始める。この条件により、削除された経路の場所を探す回数は削減され、用意された配列を無駄にすることがない。

図 4.9: 配列を用いた双方向循環リストのイメージ



## 更新対象の工夫

提案解法 2 では 1 回の更新で各頂点に対する経路を更新対象とするため、頂点を基準に更新することになる。頂点を基準に更新するとき、その頂点から出ていく辺に対しての更新（辺の始点を更新対象とする更新）とその頂点に入ってくる辺に対しての更新（辺の終点を更新対象とする更新）がある。それぞれの方法は参照順序が異なるため計算時間が変わってくる。コンピュータには最後に参照したものをキャッシュに溜め込み、次の参照が行われるときにキャッシュから取り出すことで参照時間が短くなることがある。参照時間が短く済むと同じ比較でも実装時間が短くなるため、参照順序によっておこるキャッシュの利用が多く行われ参照時間を短くした方が良い。そこで、それぞれの方法を実装し、計算時間の比較を行い考察を行う。

### 辺の始点を更新対象とする

辺の始点を更新対象とする場合を考える。辺の始点を更新対象としたとき更新対象となる頂点から出ていく辺に対して拡張探索を行う。このとき、頂点から出ていく辺の向かう先の頂点（辺の終点）は違うため探索した経路がパレート解となるか判断するリストは異なる。このためパレート解となるか判断するリストは次に参照するまでの時間が長くなるためキャッシュに保存しておくことが難しい。反対に更新対象とする頂点のリストを何回も参照するときはキャッシュに保存されていると考えられるため参照時間は短くなると考えられるが、更新対象となる頂点に対する経路に更新を行うためリストは 1 度しか参照しない。このため、実装時間は短くならないと考えられる。

### 辺の終点を更新対象とする

辺の終点を更新対象とする場合を考える。辺の終点を更新対象としたとき更新対象となる頂点に入ってくる辺に対して拡張探索を行う。このとき、頂点に入ってくる辺の元の頂点（辺の始点）は違うため更新対象となる経路は毎回異なりキャッシュに保存しておくことが難しい。反対に探索した経路がパレート解となるか判断するリストは辺の終点のリストのためキャッシュに溜め込んでおくことが可能で探索した経路がパレート解となるか判断するための参照時間は短くなると予想される。複数の経路にパレート解となるかの判断が必要で、その度にリストを参照するためリストの参照時間が短くなると実装時間も短くなると予想される。

## 4.5 提案解法の評価と分析

### 提案解法 1 と提案解法 2 の評価と分析

非負の多目的最短経路問題に対して、本研究で提案した解法の実験的評価を行なった。評価方法は各インスタンスに対して、本研究で提案する提案解法 1 により出力された解から得られた平均実装時間と、提案解法 2 により出力された解から得られた平均実装時間と比較するものである。経路を保存するリストや解であるかの判定などは同じものとする。また、最大削減率を求めるため、重みの範囲＝頂点数、最適化目的の数＝3 とし、各目的関数間の相関は弱いものとする。

尺度 1：多目的最短経路問題に対する提案解法 1 に用いた、完全多項式時間近似スキームに基づいた解の格納方法による計算時間の比較。

図 4.10 より、頂点数 1000、重みの範囲 1000 のとき、改良前の計算時間は、172,615 ms、改良後の計算時間は、20,307 ms であった。つまり、完全多項式時間近似スキームに基づいた解の格納方法の改良により、約 88% の計算時間の削減を可能にした。頂点数の増加に伴い、格納方法による改良前と改良後の計算時間の差は拡大している。そのため、頂点数をさらに増やすことで削減率も上がると考えられる。

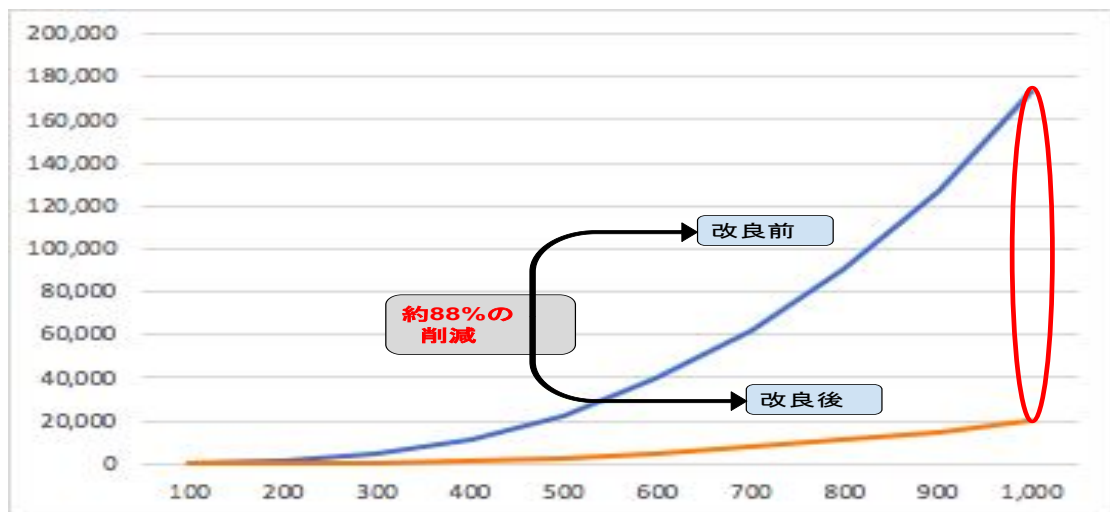


図 4.10: 多目的最短経路問題に対する提案解法 1 に用いた、完全多項式時間近似スキームに基づいた解の格納方法による計算時間の比較。表 A1 参照

尺度 2：多目的最短経路問題に対する提案解法 1 に用いた、ラベル付けアルゴリズムに基づいた解の格納方法による計算時間の比較。

図 4.11 より，頂点数 100，重みの範囲 100 のとき，改良前の計算時間は，1,654,873 ms，改良後の計算時間は，28 ms であった．つまり，ラベル付けアルゴリズムに基づいた解の格納方法の改良により，約 99% の計算時間の削減を可能にした．頂点数の増加に伴い，格納方法による改良前と改良後の計算時間の差は拡大している．そのため，頂点数をさらに増やすことで削減率も上がると考えられる．

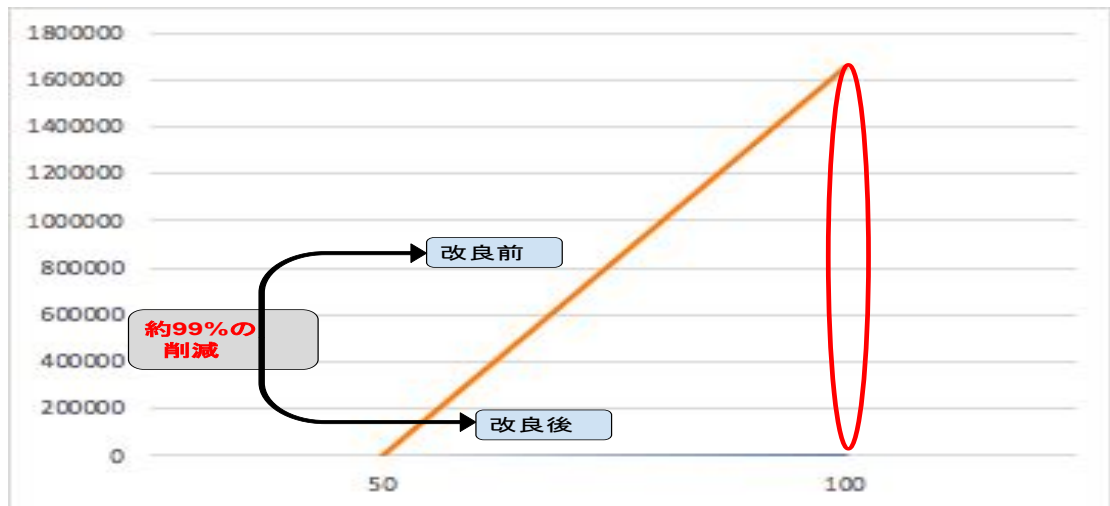


図 4.11: 多目的最短経路問題に対する提案解法 1 に用いた，ラベル付けアルゴリズムに基づいた解の格納方法による計算時間の比較．表 A2 参照

尺度 3: 多目的最短経路問題に対する提案解法 2 に用いた，動的計画法に基づいた探索順序による計算時間の比較．

図 4.12 より，頂点数 1000，重みの範囲 1000 のとき，改良前の計算時間は，20,307 ms，改良後の計算時間は，11,766 ms であった．つまり，探索順序の改良により，約 42% の計算時間の削減を可能にした．この結果に対する考察を行う．提案解法 2 の場合，探索順序を変更することによって無駄な探索を減らし探索にかかる時間を削減しているが，探索順序を変更するためにラベルを多く作っているため全体に対する探索の終了後にラベル内の解を移動させる操作が多くなる．（提案解法 1 では，更新対象とするもの → 更新対象としないもの．提案解法 2 では，更新対象とするもの → 更新対象としないもの，次回更新対象とするもの → 更新対象とするもの．）ここで，相関係数と計算時間の関係を見ていくと，相関が大きくなるにつれて解の数は少なくなり，計算時間が短くなっている．目的関数の相関と解の数の関係より，相関が大きくなるにつれて解の数は少なくなるため削除される解の数も少なくなる．

削除される解の数が少なくなると提案解法 2 の優位性が低くなる。結果より，無駄な探索をしないことによる時間の削減<ラベル内の解を移動させる操作にかかる時間となっていると考えられる。頂点数の増加に伴い，格納方法による改良前と改良後の計算時間の差は拡大している。そのため，頂点数をさらに増やすことで削減率も上がると考えられる。

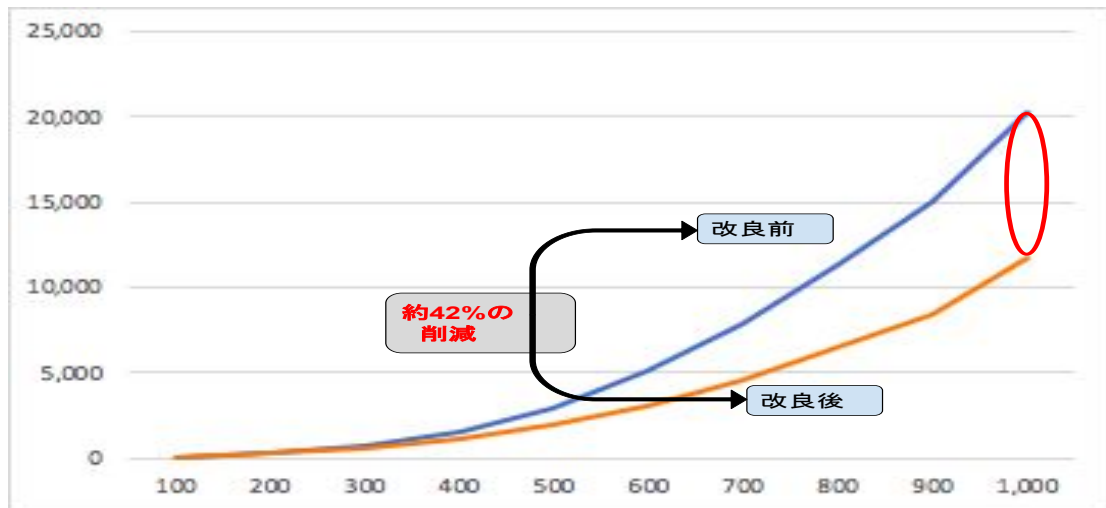


図 4.12: 多目的最短経路問題に対する提案解法 2 に用いた，動的計画法に基づいた探索順序による計算時間の比較。表 A3 参照

尺度 4: 多目的最短経路問題に対する提案解法 1 と提案解法 2 において，各目的関数の相関係数による計算時間の比較。

図 4.13 より，頂点数 1000，重みの範囲 1000 のとき，目的関数の相関係数の値が約 0.90 のときに提案解法 1 の計算時間が提案解法 2 の実装時間より短くなっている。つまり，目的関数の相関係数の値が約 0.90 のときに計算時間が逆転している。この結果に対する考察を行う。提案解法 2 の場合，探索順序を変更することによって無駄な探索を減らし探索にかかる時間を削減しているが，探索順序を変更するためにラベルを多く作っているため全体に対する探索の終了後にラベル内の解を移動させる操作が多くなる。(提案解法 1 では，更新対象とするもの → 更新対象としないもの。提案解法 2 では，更新対象とするもの → 更新対象としないもの，次回更新対象とするもの → 更新対象とするもの。) ここで，相関係数と計算時間の関係を見ていくと，相関が大きくなるにつれて解の数は少なくなり，計算時間が短くなっている。目的関数の相関と解の数の関係より，相関が大きくなるにつれて解の数は少



なくなるため削除される解の数も少なくなる．削除される解の数が少なくなると提案解法 2 の優位性が低くなる．結果より，目的関数の相関係数の値が約 0.90 以上のとき，無駄な探索をしないことによる時間の削減＜ラベル内の解を移動させる操作にかかる時間となっていると考えられる．図 4.13 は頂点に入ってくる辺の重みベクトルに対しての相関を求めているが，頂点から出ていく辺の重みベクトルに対しての相関でも同じことが言える．(表 A7 参照) 頂点に入ってくる辺の重みベクトルに対しての相関と頂点から出ていく辺の重みベクトルに対しての相関の間には関係が存在しないため，両方の相関が低い場合のみ，無駄な探索をしないことによる時間の削減＞ラベル内の解を移動させる操作にかかる時間となる．

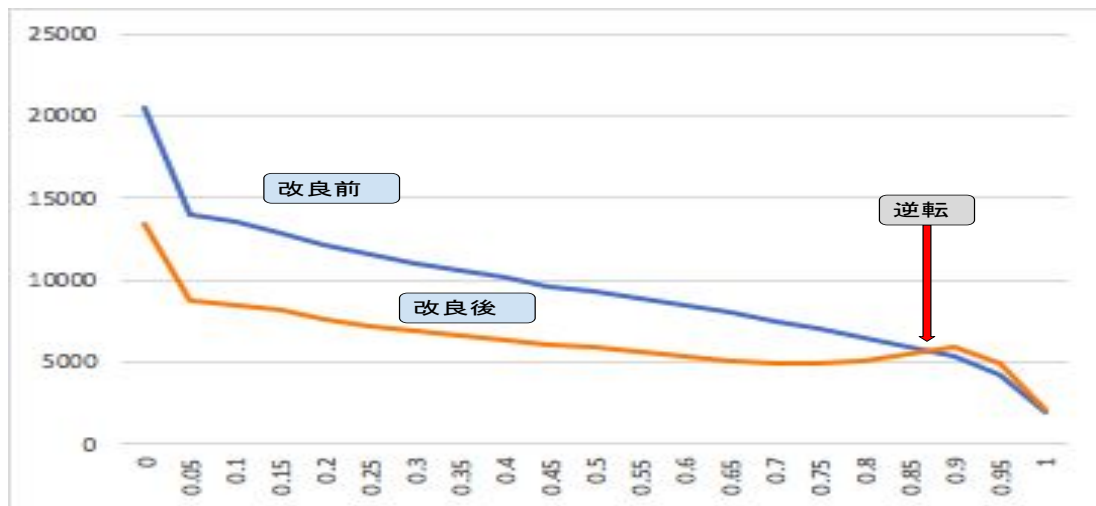


図 4.13: 多目的最短経路問題に対する提案解法 1 と提案解法 2 において，各目的関数の相関係数による計算時間の比較．表 A6 参照

#### 負の重みを考慮した解法の評価と分析

実数（負の重みを含む）の多目的最短経路問題に対して，本研究で提案した解法の実験的評価を行なった．評価方法は各インスタンスに対して，本研究で提案する 3 つの解法により出力された解から得られた平均実装時間を比較するものである．負のサイクルが存在しない場合を考慮するために非負の場合も比較する．また，最大削減率を求めるため，重みの範囲＝頂点数，最適化目的の数＝3 とし，各目的関数間の相関は弱いものとする．負のサイクルが存在するときの比較のため 3 つの目的関数のうち 1 つの目的関数に負のサイクルが存在している．

尺度 1：負のサイクルが存在しない多目的最短経路問題に対する負の重みを考慮した解法による計算時間の比較.

表 A8 より，頂点数 1000，重みの範囲 1000 のとき，負の閉路を先に検出したときの計算時間は，12,921 ms，探索途中で負の閉路を検出したときの計算時間は，12,683 ms，負の閉路検出で得られた解を基に探索したときの計算時間は，28,756 ms であった．つまり，負の閉路を先に検出した場合と探索途中で負の閉路を検出した場合の計算時間がもっとも短いということになる．この結果に対する考察を行う．負の閉路を先に検出した場合と探索途中で負の閉路を検出した場合，負の閉路検出に多少の時間はかかるが探索順序は提案解法 2 と同じになる．しかし，負の閉路検出で得られた解を基に探索した場合の探索順序は提案解法 2 と異なる．これは，探索開始時の更新対象となる解が多く存在しているためである．負の閉路検出で得られた解は経由する頂点が複数である経路であるため，負の閉路検出で得られた解を基にして探索したときに得られる解は後に他の解に支配され削除される可能性が高い．そのため，非効率であると考えられる．頂点数の増加に伴い，格納方法による改良前と改良後の計算時間の差は拡大している．そのため，頂点数をさらに増やすことでそれぞれに対する効率の差が大きくなると考えられる．

尺度 2：負のサイクルが存在する多目的最短経路問題に対する負の重みを考慮した解法による計算時間の比較.

表 A9 より，頂点数 1000，重みの範囲 1000 のとき，負の閉路を先に検出したときの計算時間は，2,029 ms，探索途中で負の閉路を検出したときの計算時間は，1,569 ms，負の閉路検出で得られた解を基に探索したときの計算時間は，2,898 ms であった．つまり，探索途中で負の閉路を検出した場合の計算時間がもっとも短いということになる．この結果に対する考察を行う．負の閉路を先に検出した場合と探索途中で負の閉路を検出した場合，負の閉路検出に多少の時間はかかるが探索順序は提案解法 2 と同じになる．しかし，負の閉路検出で得られた解を基に探索した場合の探索順序は提案解法 2 と異なる．これは，探索開始時の更新対象となる解が多く存在しているためである．負の閉路検出で得られた解は経由する頂点が複数である path であるため，負の閉路検出で得られた解を基にして探索したときに得られる解は後に他の解に支配され削除される可能性が高い．そのため，非効率であると考えられる．また，今回の実験ではすぐに負の閉路が見つかる場合のため探索途中で負の閉路を検出したとき最短で 2 回目の探索で負の閉路を発見できる．負の閉路を先に検



出した場合はそれぞれの目的関数に対して  $O(VE)$  の計算時間がかかるため時間がかかる。頂点数の増加に伴い、格納方法による改良前と改良後の計算時間の差は拡大している。そのため、頂点数をさらに増やすことでそれぞれに対する効率の差が大きくなると考えられる。すぐに負の閉路が見つからない場合、探索途中で負の閉路を検出したときの方が負の閉路を先に検出した場合よりも多くの計算時間を必要とし、負の閉路発見後も目的関数の削除におけるステップ数が多いため、負の閉路を先に検出した方が効率的になると考えられる。

## 工夫に対する評価と分析

非負の多目的最短経路問題に対して、本研究で提案した解法の実験的評価を行なった。評価方法は各インスタンスに対して、本研究で提案する提案解法により出力された解から得られた平均実装時間と、実装における工夫を考慮しない提案解法 2 により出力された解から得られた平均実装時間と比較するものである。また、最大削減率を求めるため、重みの範囲＝頂点数、最適化目的の数＝3 とし、各目的関数間の相関は弱いものとする。

尺度 1：多目的最短経路問題に対する提案解法 2 に用いた、リスト構造による計算時間の比較。

表 A10 より、頂点数 500、重みの範囲 500 のとき、ArrayList を使用したときの計算時間は、2,913 ms、配列を用いた双方向リストの計算時間は、2,037 ms、class を用いた双方向リストの計算時間は、1,789 ms であった。つまり、class を用いた双方向リストの計算時間がもっとも短いということになる。この結果に対する考察を行う。ArrayList を使用した場合、双方向リストは使用できないため経路の追加/削除、ラベルからラベルへの経路の移動にかかる計算時間が長くなるため全体の実装時間が長くなったと考えられる。逆に、双方向リストを使用した場合、経路の追加/削除、ラベルからラベルへの経路の移動にかかる計算時間が短くなるため全体の実装時間が短くなったと考えられる。class を使用した場合、経路が増えるたびに新しい変数を作成するため計算時間がかかる。配列を用いた場合、メモリ削減のため空の要素を管理し経路の保存場所を決定するため計算時間がかかる。経路の数が大きくなるほど空の管理は難しくなるため、インスタンスが大きくなるにつれて非効率になると考えられる。頂点数の増加に伴い、格納方法による改良前と改良後の計算時間の差は拡大している。そのため、頂点数をさらに増やすことでそれぞれに対する効率の差が大きくなると考えられる。

尺度 2：多目的最短経路問題に対する提案解法 2 に用いた，更新対象による計算時間の比較．

表 A11 より，頂点数 500，重みの範囲 500 のとき，始点を更新対象としたときの計算時間は，終点を更新対象としたときの計算時間は，1,789 ms であった．つまり，終点を更新対象としたときの計算時間が始点を更新対象としたときの計算時間より短いということになる．この結果に対する考察を行う．辺の始点を更新対象とする場合を考える．辺の始点を更新対象としたとき頂点から出ていく辺の向かう先の頂点（辺の終点）は違うため，パレート解となるか判断するリストは次に参照するまでの時間が長くなると予想されキャッシュに保存しておくことが難しい．反対に辺の終点を更新対象としたとき探索した経路がパレート解となるか判断するリストは辺の終点のリストのためキャッシュに溜め込んでおくことが可能で探索した経路がパレート解となるか判断するための参照時間は短くなると予想される．複数の経路にパレート解となるかの判断が必要で，その度にリストを参照するためリストの参照時間が短くなると実装時間も短くなると予想される．頂点数の増加に伴い，格納方法による改良前と改良後の計算時間の差は拡大している．そのため，頂点数をさらに増やすことでそれぞれに対する効率の差が大きくなると考えられる．

## 第5章 結論

### 5.1 研究成果

成果 1：非負における多目的最短経路問題に対して，ベルマンフォード法の探索順序およびデータの格納に改良を加えた解法を提案した．

多目的最短経路問題に対して，負の重みを許しているので，ダイクストラ法からベルマンフォード法に移行し，多目的への拡張を行った．格納されているデータは経路のため，更新対象や各目的関数における比較は経路に対して行われる．そこで，必要な経路を発見しやすいデータ構造を提案した．提案したデータ構造により，頂点数  $n$ ，最適化目的の数  $k$  のとき，次の成果が得られた．パレート解であるかの判定に要する計算量を  $O(k * n^n)$  から  $O(k * n^{n-1})$  に削減し，計算時間を最大 99% 削減した．全体の探索における経路の更新回数に対する合計計算量を  $O(n^{n+1})$  から  $O(n^n)$  に削減し，計算時間を最大 88% 削減した．経由する頂点数が少ない経路から順に探索されるよう探索順序の指定を行なった．また，全体の最大計算量を  $O(k * n^{2n+2})$  から  $O(k * n^{2n})$  に削減した．一般に経由する頂点数が少ない経路のほうが最短経路になりやすいため，探索順序の変更により無駄な更新が行われる回数を削減し，計算時間を最大 42% 削減した．

以上の改良より，従来研究のもっとも実装時間の短い解法と比較した結果，同じインスタンスに対して計算時間を最大 93% 削減した．

成果 2：負のサイクルが存在する場合を考慮した解法を提案した．

多目的最短経路問題において負のサイクルが存在する場合，負のサイクルが存在しない目的関数のみでのパレート解と定義し，負のサイクルの検出を行う解法を提案した．以上より，多目的最短経路問題における負のサイクルの検出を行うことで負の重みを考慮し，扱えるインスタンスの範囲を広げた．

まとめ：本研究では最適化目的の数 = 3 である単一始点最短経路問題に対する解法を提案した．しかし，最適化目的の数が 3 より多い場合の検証は行えておらず全点对最短

経路問題に対する解法は提案できていない。また、解法全体の評価はされていないが実装時間はインスタンスの大きさ、解の大きさによって変化すると考えられるため、それらの要素と実装時間の上限にどのような関係があるか求めたい。

## 5.2 今後の課題

以下に、本研究で明らかにできなかった課題を今後の課題としてまとめる。

**課題 1：**多目的全点对最短経路問題に対する解法の提案。

本研究では、1つの始点からその他の頂点への最短経路を求める多目的単一始点最短経路問題についての解法を行なった。しかし、最短経路問題には、任意の頂点間の最短経路を求める全点对最短経路問題が存在するため多目的最適化問題においても考慮すべきであるが、効率的な解法は提案されていない。

**課題 2：**インスタンスの大規模化。

本研究では、1つの計算機での実装を行うための解法を提案したが、1つの計算機での計算には限界がある。そこで並列計算手法を考える。複数の計算機を通信を用いて使用することにより、より大規模なインスタンスが実装可能になることが考えられる。

**課題 3：**計算時間の評価。

本研究では、1つの計算機での実装を行うための解法を提案し、一部のステップにおける計算時間を評価し、実装でも優位性を確かめた。しかし、解法全体の評価はされていない。実装時間はインスタンスの大きさ、解の大きさによって変化すると考えられるため、それらの要素と実装時間の上限にどのような関係があるか求めたい。この関係を求めることにより、実装前にどれだけの時間がかかるかの予測や、実装時にどれだけ効率的であるかを求めることができると考えられる。また、一部のステップにおける計算時間を評価は指数時間となっているため、多項式時間での評価や、次数が低くなるような評価をしたい。

## 参考文献

- [1] Abdullah Konaka, David W.Coitb, A.E.: Multi-objective optimization using genetic algorithms: A tutorial, *Reliability Engineering & System Safety*, vol. 91, pp. 992–1007. Elsevier (2006)
- [2] Ahuja, R., Magnanti, T., Orlin, J.: *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, N.J. : Prentice-Hall (1993)
- [3] AT, M.T., T Labs Research, Florham Park, N.: Undirected single-source shortest paths with positive integer weights in linear time, *Journal of the ACM*, vol. 46, pp. 362–394. Association for Computing Machinery (1999)
- [4] Breugem, T., Dollevoet, T., van den Heuvel, W.: Analysis of FPTASes for the multi-objective shortest path problem, *Computers & Operations Research*, vol. 78, pp. 44–58. Elsevier B.V. (2017)
- [5] Floyd, R.W.: Algorithm 97: Shortest Path, *Communications of the ACM*, vol. 5, p. 345. Association for Computing Machinery (1962)
- [6] N.Takahashi, H.Yamamoto, T.Akiba: Algorithm for Obtaining Optimal Path in Three-objective Network with a Reduction of Search Spac, *Innovation and Supply Chain Management*, vol. 9, pp. 103–109. Japan Management Training Center (2015)
- [7] R, B.: On a Routing Problem, *QuarterlyofApplied Mathematics*, vol. 16, pp. 87–90. American Mathematical Society (1958)
- [8] Santos, J.L.E.D., Paixo, J.: Labeling Methods for the General Case of the Multi-objective Shortest Path Problem A Computational Study, *Intelligent Systems, Control and Automation: Science and Engineering*, vol. 61, pp. 489–502. Springer (2012)
- [9] W, D.E.: A note on two problems in connection with graphs, *Numerische Mathematik*, vol. 1, pp. 269–271. Springer (1959)

## 付 録 A 多目的最短経路問題に対する解法の実験結果

表 A.1: 頂点数と重みの範囲に対する完全多項式時間近似スキームに基づいた解の格納方法の計算時間（単位：ms）の比較（最適化目的の数：3，各目的関数間に相関なし），実験回数：約 10 万回

頂点数＝重みの範囲	解の数	改善前	改良後
100	6,121	459	90
200	19,939	1,804	322
300	38,807	4,881	731
400	61,338	11,687	1,620
500	87,141	22,727	2,997
600	115,703	39,786	5,139
700	146,406	61,968	7,938
800	179,819	90,555	11,265
900	214,839	126,755	15,092
1,000	251,852	172,615	20,307

表 A.2: 頂点数と重みの範囲に対するラベル付けアルゴリズムに基づいた解の格納方法の計算時間（単位：ms）の比較（最適化目的の数：3，各目的関数間に相関なし），実験回数：60 回

頂点数＝重みの範囲	改善前	改良後
50	439	3
100	1,654,873	28

表 A.3: 頂点数と重みの範囲に対する探索順序に基づいた解の格納方法の計算時間（単位：ms）の比較（最適化目的の数：3，各目的関数間に相関なし），実験回数：約 10 万回

頂点数＝重みの範囲	解の数	改善前	改良後
100	6,121	90	83
200	19,939	322	274
300	38,807	731	565
400	61,338	1,620	1,117
500	87,141	2,997	1,938
600	115,703	5,139	3,135
700	146,406	7,938	4,577
800	179,819	11,265	6,457
900	214,839	15,092	8,465
1,000	251,852	20,307	11,766

表 A.4: 頂点数と重みの範囲に対する探索順序に基づいた解の格納方法の削除される解の数の比較（最適化目的の数：3，各目的関数間に相関なし），実験回数：約 10 万回

頂点数＝重みの範囲	解の数	従来研究	改善前	改良後
100	6,121	5,136	5,136	2,606
200	19,939	22,009	22,009	10,528
300	38,807	49,320	49,320	22,814
400	61,338	85,646	85,646	38,806
500	87,141	130,495	130,495	58,133
600	115,703	182,761	182,761	80,425
700	146,406	242,159	242,159	105,387
800	179,819	308,104	308,104	133,120
900	214,839	380,748	380,748	162,784
1,000	251,852	459,256	459,256	195,413

表 A.5: 頂点数と重みの範囲に対する実行可能解の計算時間（単位：ms）の比較（最適化目的の数：3，各目的関数間に相関なし），実験回数：3回

頂点数＝重みの範囲	解の数	従来研究	改善前	改良後
100	6,319	170	33	28
200	21,249	1,666	225	180
300	43,742	5,383	625	561
400	66,356	11,335	1,205	1,030
500	92,661	20,369	2,465	1,766
600	127,998	37,682	3,881	2,860
700	154,392	62,498	5,860	3,879
800	190,454	72,925	7,875	5,593
900	236,023	109,416	11,270	8,112
1,000	270,261	173,189	15,043	10,427
1,100	277,007	192,805	21,598	13,824
1,200	324,186	247,645	23,965	17,707
1,300	366,179	199,785	34,085	21,578
1,400	423,442	356,512	52,437	26,657
1,500	428,851	409,573	47,912	32,219
1,600	499,582	525,150	61,696	38,498
1,700	531,096	648,257	70,175	43,645
1,800	596,137	652,252	83,488	51,557
1,900	641,731	1,076,189	92,975	60,735
2,000	741,472	2,597,995	180,264	88,090
3,000	1,215,694	6,936,107	266,882	205,562
4,000	1,870,729	8,213,518	489,084	377,354
5,000	2,512,844	12,034,818	1,593,567	730,709
6,000	3,531,481	21,577,070	1,689,733	1,076,165
7,000	4,041,990	28,203,255	2,193,079	1,659,099
8,000	5,006,335	31,254,881	3,197,729	2,131,159
9,000	5,944,410	57,888,144	3,834,144	2,573,077
10,000	6,995,929	65,294,657	5,537,949	3,768,780
11,000	8,055,357	$\infty$	8,501,472	6,059,436
12,000	9,332,482	$\infty$	9,193,497	6,143,881
13,000	10,604,845	$\infty$	11,402,847	6,774,594
14,000	11,418,667	$\infty$	13,897,933	8,560,172
15,000	12,244,607	$\infty$	15,008,922	10,019,615
16,000	$\infty$	$\infty$	$\infty$	$\infty$



表 A.6: 頂点に対する入力辺の相関係数ごとの計算時間（単位：ms）と削除回数の比較  
（頂点数：1000, 重みの幅：1000, 最適化目的の数：3）, 実験回数：約 2 万回

相関係数	解の数	改善前計算時間	改善後計算時間	改善前削除数	改善後削除数
0	246,955	20,506	13,478	447,440	186,749
0.05	185,662	13,962	8,811,	420,840	166,948
0.1	161,191	13,487	8,492	315,199	140,244
0.15	172,748	12,871	8,120	312,996	150,422
0.2	161,141	12,189	7,669	324,362	143,351
0.25	134,353	11,539	7,243	313,794	132,049
0.3	145,781	10,981	6,854	332,715	128,478
0.35	141,302	10,511	6,580	287,743	129,807
0.4	151,405	10,130	6,346	288,121	123,322
0.45	142,391	9,582	6,014	293,642	129,536
0.5	132,419	9,293	5,865	267,527	108,954
0.55	127,512	8,944	5,581	258,366	110,932
0.6	118,846	8,410	5,360	292,424	123,085
0.65	109,101	8,021	5,117	216,053	91,999
0.7	122,610	7,430	4,952	239,031	96,521
0.75	112,932	7,035	4,885	215,665	98,276
0.8	103,793	6,480	5,120	187,216	93,555
0.85	111,638	5,905	5,459	186,635	84,965
0.9	91,323	5,304	5,866	156,423	73,575
0.95	75,277	4,173	4,966	132,447	62,707
1	55,699	1,930	2,066	87,051	42,773

表 A.7: 頂点に対する出力辺の相関係数ごとの計算時間（単位：ms）と削除回数の比較（頂点数：1000, 重みの幅：1000, 最適化目的の数：3），実験回数：約 2 万回

相関係数	解の数	改善前計算時間	改善後計算時間	改善前削除数	改善後削除数
0	257,756	19,865	13,254	476,565	204,555
0.05	180,432	13,794	8,846	372,821	155,101
0.1	184,311	13,279	8,536	372,142	156,539
0.15	173,748	12,653	8,126	341,195	153,053
0.2	153,605	11,942	7,683	350,808	144,610
0.25	162,542	11,424	7,296	339,908	138,337
0.3	152,887	10,942	6,952	288,288	126,227
0.35	146,081	11,297	7,213	309,543	134,043
0.4	145,310	10,106	6,438	286,549	119,165
0.45	130,655	9,598	6,157	263,506	109,500
0.5	126,388	9,180	5,824	243,431	105,708
0.55	131,804	8,775	5,621	251,082	111,571
0.6	127,963	8,300	5,321	260,390	112,661
0.65	135,237	7,892	5,122	267,342	110,063
0.7	118,104	7,365	4,813	253,141	107,362
0.75	127,453	6,842	4,630	217,825	96,583
0.8	100,776	6,312	4,668	197,383	88,598
0.85	99,316	5,781	5,048	182,147	85,528
0.9	90,367	5,073	5,266	165,269	75,060
0.95	78,995	4,056	4,695	125,484	63,460
1	52,036	1,914	2,103	73,916	43,110

表 A.8: 非負の頂点数と重みの範囲に対する負考慮解法ごとの計算時間（単位：ms）の比較（最適化目的の数：3, 各目的関数間に相関なし），実験回数：3000 回

頂点数＝重みの範囲	解の数	負の閉路を先に検出	探索途中で検出	得られた解を基に探索
600	118,123	2,813	2,752	5,811
700	150,623	4,289	4,255	9,209
800	183,006	5,742	5,551	12,114
900	215,938	7,766	7,564	16,631
1,000	251,326	12,921	12,683	28,757

表 A.9: 負の重みを含む（1つの目的関数に負の閉路が存在，すぐに閉路が見つかる）頂点数と重みの範囲に対する負考慮解法ごとの計算時間（単位：ms）の比較（最適化目的の数：3, 各目的関数間に相関なし），実験回数：3000 回

頂点数＝重みの範囲	解の数	負の閉路を先に検出	探索途中で検出	得られた解を基に探索
600	14,742	568	491	856
700	17,706	798	683	1,196
800	21,806	1,151	986	1,690
900	24,442	1,466	1,186	2,217
1,000	29,051	2,029	1,569	2,898

表 A.10: 頂点数と重みの範囲に対するリストの工夫ごとの計算時間（単位：ms）の比較（最適化目的の数：3，各目的関数間に相関なし），実験回数：4000 回

頂点数＝重みの範囲	解の数	双方向循環リスト	ArrayList	配列管理
100	6,054	28	35	29
200	20,630	201	318	205
300	38,758	566	1,158	572
400	63,646	1,161	2,773	1,176
500	86,275	1,790	4,780	2,037

表 A.11: 頂点数と重みの範囲に対する更新対象の工夫ごとの計算時間（単位：ms）の比較（最適化目的の数：3，各目的関数間に相関なし），実験回数：2000 回

頂点数＝重みの範囲	解の数	辺の終点を更新対象	辺の始点を更新対象
100	6,054	28	51
200	20,630	201	321
300	38,758	566	825
400	63,646	1,161	1,790
500	86,275	1,790	2,914

表 A.12: 頂点数ごとの計算時間（単位：ms）の比較（重みの幅：1000，最適化目的の数：3，各目的関数間に相関なし），実験回数：約 3000 回

頂点数	改善前計算時間	改善後計算時間	従来計算時間
100	197	168	5,726
200	1,152	907	20,033
300	3,756	2,982	39,113
400	7,312	5,819	61,123
500	11,227	7,456	85,744
600	11,554	6,928	116,011
700	12,901	8,515	144,736
800	23,644	14,779	174,663
900	22,201	14,093	213,478
1,000	29,744	18,544	251,971

表 A.13: 重みの範囲ごとの計算時間（単位：ms）の比較（頂点数：1000，最適化目的の数：3，各目的関数間に相関なし），実験回数：約 3000 回

重みの範囲	改善前計算時間	改善後計算時間	従来計算時間
100	46,677	28,502	208,974
200	37,146	24,333	239,559
300	30,315	19,921	248,892
400	30,526	19,639	259,125
500	28,976	19,166	254,776
600	28,683	19,561	255,832
700	28,175	19,157	253,396
800	28,272	19,296	252,128
900	27,813	19,190	252,658
1,000	28,249	18,766	249,359
1,100	27,392	18,712	248,251
1,200	29,003	19,437	255,863
1,300	27,351	18,940	251,286
1,400	27,474	18,787	251,007
1,500	28,001	18,770	249,709

表 A.14: 最適化目的の数ごとの計算時間（単位：ms）の比較（頂点数=重みの範囲=1000，各目的関数間に相関なし），実験回数：約 1000 回

最適化目的の数	解の数	改善前計算時間	改善後計算時間
1	1,000	1,104	2,516
2	29,764	10,645	9,972
3	253,189	65,963	45,505
4	1,253,387	294,329	133,776
5	4,096,826	1,278,071	713,449

表 A.15: 従来研究に対する頂点数と重みの範囲ごとの計算時間（単位：ms）の比較（最適化目的の数：3，各目的関数間に相関なし），実験回数：60 回

頂点数=重みの範囲	拡張ダイクストラ	完全多項式時間近似スキーム	ラベル付けアルゴリズム
50	1,617	439	18
100	1,476,912	1,654,873	142

# 謝辞

本研究を進めるにあたり，指導教員の宋 少秋教授には研究に対する助言や熱心な指導をしていただきましたことを心から感謝いたします。またゼミや日常で多くの知識や示唆をいただいた研究室の先輩，同期の方々に感謝いたします。

2019 年 1 月 31 日 中野 壱帥

卒業論文  
多目的最短経路問題のための  
拡張ベルマンフォード法の提案

中野 志帥