Vol.2018-AL-168 No.7
2018/5/26

有向道路ネットワーク向け最短路クエリのための効率的枝刈り探索手法

小池 敦[1,a)]

概要：道路ネットワーク上の最短路クエリのための新しいラベリング手法を提案する．Pruned Highway Labeling は枝刈りダイクストラ法を活用した効率的な前処理が特徴である．しかし，無向グラフでしか使用することができない．本論文では有向グラフに対応するために新しい枝刈り探索手法を提案する．アルゴリズムの正当性について証明を行い，実装したプログラムの基礎的な評価を行う．

キーワード：グラフアルゴリズム，経路探索クエリ，道路ネットワーク

Efficient Pruned Searches for Shortest Path Queries in Directed Road Networks

Atsushi Koike[1,a)]

Abstract: We propose a new labeling method for shortest path queries in road networks. Pruned Highway Labeling can perform efficient preprocessing utilizing pruned Dijkstra searches. However, it can be used only in undirected graphs. We propose a new pruned search algorithm for directed graphs. We prove the correctness of our algorithm and provide a primary evaluation of our program.

Keywords: graph algorithms, route search queries, road networks

1. はじめに

経路探索は道路ネットワークにおいて最も基本的な処理の一つであり多くの研究が行われている [1]．経路探索のための古典的なアルゴリズムとしてダイクストラ法があり，その基本的な改良手法として，双方向ダイクストラ法や A* アルゴリズム [2]，双方向 A*探索 [3] などが提案されている．自動車向けの経路探索では交差点における右左折コストの考慮が必要となるが，グラフをライングラフに変換することでダイクストラ法を用いた最短経路探索が可能となることが知られている [4], [5]．また，リンクコストが時刻に依存する場合でも，グラフが FIFO プロパティを満たすならばダイクストラ法を用いた最短経路探索が可能であることも知られている [6]．道路ネットワークにおける渋滞考慮探索を行う場合 FIFO プロパティを満たしており，また公共交通網において時刻表探索を行う場合もグラフに適切な変換を行うことで FIFO プロパティを満たすことができるようになる [6]．ダイクストラ法以外の経路探索手法として，ベルマンフォードアルゴリズム [7], [8] がある．計算量はダイクストラ法よりも大きくなるものの，ダイクストラ法より並列化が容易である点や負のコストのリンクを扱える点が長所である．並列化については，ベルマンフォードアルゴリズムとダイクストラ法を混合したアルゴリズムとしてデルタステッピングアルゴリズム [9] が提案されており，大型計算機でこれを高速実行するための手法 [10] も提案されている．

経路探索を高速化する手法の一つとして，事前にグラフに対して前処理を行うことで，経路探索クエリを高速化する手法がある．古典的な手法として，ALT アルゴリズム [11] や Reach[12] があげられる．ALT アルゴリズムは事前処理

[1] 一関工業高等専門学校
[a)] koike@ichinoseki.ac.jp

を行うことで A*コストをより高精度に見積もることができるようにする手法である．Reach は事前に各頂点の重要度を計算することでダイクストラ法において効率的な枝刈りをする手法である．また，Contraction Hierarchy[13] や Transit Node Routing[14] も前処理により経路探索クエリを高速化する手法である．これらは前処理を活用してグラフ探索を高速化するアプローチである．グラフ探索を行わずに経路を求めるアプローチとして，ラベリング法がある．そのうち Hub Labeling[15] は現状最高速の経路探索クエリを実現する手法である．しかし，前処理計算時間が大きくなることが欠点である．

Pruned Highway Labeling (PHL)[16] は，ラベリング法の一種であるが，前処理の高速実行が特徴である．本手法では前処理に枝刈りダイクストラ法 [17] を使うことで，効率的な前処理を可能としている．本手法では，グラフを複数の最短経路に分解することで効率的なクエリを実現するが，この処理は無向グラフに対してしか適用することができないという欠点がある．道路ネットワークでは一般に逆向きの道路はコストが異なっており，また多くの一方通行道路があるため，道路ネットワークを無向グラフで表現することはできない．

本論文では有向グラフに対するラベリング法を提案する．PHL と同様に枝刈り探索を活用した効率的な前処理を行う．まず，2 節で定式化等の準備をしたのち，3 節で提案アルゴリズムの詳細を述べ，4 節で使用したヒューリスティクスについて述べる．そして，5 節で米国道路ネットワークを用いた基礎的な評価実験を行う．最後に 6 節でまとめを行う．

2. 準備

2.1 定式化

本論文では，道路上の経路探索を考える．入力グラフは有向グラフであり，各辺は正重みのコストを持っている．本論文ではノードはコストを持たない．本論文で扱う最短経路問題は以下のように定式化される．

(1) 前処理フェーズ

入力: 正重み有向グラフ $G = (V, E)$
出力: 前処理データ D

(2) クエリフェーズ

入力: G, D, 出発地 $s \in V$, 目的地 $t \in V$
出力: s から t への最短経路のコスト

性能評価基準として，クエリ応答時間，前処理計算時間，前処理データサイズを用いる．以後，ノード s からノード t への最短経路のコストを $d(s,t)$ と記述することにする．有向グラフ上の最短経路を考えているため，一般には $d(s,t)$ と $d(t,s)$ は等しくならない．また，本論文ではグラフは強連結であると仮定して説明するが，提案アルゴリズムは強連結でないグラフを用いる場合に拡張することも可能である．

2.2 ラベリング法

提案アルゴリズムはラベリング法と呼ばれるアプローチを用いている．そこで本節では，標準的なラベリング法について概要を説明する．以下では有向グラフ G のノード s から t への最短経路を s-t 最短経路と呼び，s-t 最短経路により構成される部分グラフを $P_{s,t} = (V_{s,t}, E_{s,t})$ と書く．s-t 最短経路が複数存在する場合，$P_{s,t}$ はそのうちの一つである．一般には，$P_{s,t}$ と $P_{t,s}$ は等しくならない．

s-t 最短経路の少なくとも一つがノード w を通過するとき，w は s-t 最短経路をカバーするという．すなわち，ノード s,t に対し，ある s-t 最短経路 $P_{s,t} = (V_{s,t}, E_{s,t})$ が存在し，$w \in V_{s,t}$ を満たすとき，w は s-t 最短経路をカバーする．

このとき，s-t 最短経路コストは $d(s,w) + d(w,t)$ である．ノード s にペア $(w, d(s,w))$ を保持し，ノード t にペア $(w, d(w,t))$ を保持することにすると，w が s-t 最短経路をカバーすることがわかっている場合には，s-t 最短経路コストは $d(s,w) + d(w,t)$ と即座に求まる．

次に，$G = (V, E)$ のノードの部分集合 $V_s \subseteq V, V_t \subseteq V$ に対し，V_s 中のノードから V_t 中のノードへのすべての最短経路をカバーすることを考える．任意の $s \in V_s, t \in V_t$ に対し，あるノード $w \in V_w$ が存在し，w が s-t 最短経路をカバーする時，V_w は V_s から V_t への最短経路をカバーするという．この時，s-t 最短経路コストは以下のように算出できる．

$$d(s,t) = \min_{w \in V_w} \{d(s,w) + d(t,w)\}$$

よって，V_s, V_t 中のそれぞれのノードが，V_w 中の全ノードへのコストを保持することにより s-t 最短経路コストを高速に算出することができる．

しかし，全ノードへのコストを保持することは実際には不要であり，不要なノードへのコストの保持をやめることにより，データサイズを削減することができる．V_s 中の各ノード s は V_w の部分集合 $H_0(s) \subset V_w$ 中の各ノードへのコストを保持することとし，V_t 中の各ノード t は V_w の部分集合 $H_1(t) \subset V_w$ 中の各ノードへのコストを保持することとする．ただし，$H_0(s)(s \in V_s)$,$H_1(t)(t \in V_t)$ は以下の性質を満たすものとする：任意の $s \in V_s, t \in V_t$ に対し，あるノード $w \in H_0(s) \cap H_1(t)$ が存在し，w が s-t 最短経路をカバーする．この性質を**カバープロパティ**と呼ぶ．この時，s-t 最短経路コストは以下のように算出できる．

$$d(s,t) = \min_{w \in H_0(s) \cap H_1(t)} \{d(s,w) + d(t,w)\}$$

カバープロパティを満たす $H_0(s)(s \in V_s)$,$H_1(t)(t \in V_t)$

Algorithm 1 ラベリング法による最短経路クエリ

```
1: procedure QueryUsingLabels(s, t, L_0(s), L_1(t))
2:     cost ← ∞
3:     i, j ← 0
4:     while i < |L_0(s)| かつ j < |L_1(t)| do
5:         (w_0, d_0) ← L_0(s)[i]
           ▷ L_0(s)[i] は L_0(s) の i+1 番目のペア．d_0 = d(s, w_0)
6:         (w_1, d_1) ← L_1(t)[j]
           ▷ L_1(t)[j] は L_1(t) の j+1 番目のペア．d_1 = d(w_1, t)
7:         if w_0 = w_1 then                    ▷ w_0 ∈ H_0(s) ∩ H_1(t)
8:             cost ← min{cost, d_0 + d_1}
9:             i ← i + 1, j ← j + 1
10:        else if w_0 < w_1 then
11:            i ← i + 1
12:        else if w_0 > w_1 then
13:            j ← j + 1
14:        end if
15:    end while
16:    return cost
17: end procedure
```

が得られた場合に経路探索クエリを高速実行するためのデータ構造を考える．V_s 中の各ノード s は $H_0(s)$ 中の全ノード w に対しペア $(w, d(v,w))$ を保持する．この集合を s の出発地側ラベルと呼び，$L_0(s)$ と書くことにする．$L_0(s)$ 中のペアは w の値により昇順にソートされているものとし，$d(s,w)$ については前処理フェーズで事前計算しておく．また，V_t 中の各ノード t は $H_1(t)$ 中の全ノード w に対しペア $(w, d(w,t))$ を保持する．この集合を t の目的地側ラベルと呼び，$L_1(t)$ と書くことにする．出発地側ラベルと同様，$L_1(t)$ 中のペアは w の値により昇順にソートされているものとし，$d(w,t)$ については，前処理フェーズで事前計算しておく．この時，ノード s から t への最短経路クエリアルゴリズムは Algorithm 1 のようになる．$L_0(s), L_1(t)$ がソートされていることにより，それぞれのラベルを 1 回ずつスキャンする計算量で最短経路コストが求まる．すなわち，本アルゴリズムの計算量は $\mathcal{O}(L_0(s)+L_1(t))$ である．

$V_s = V, V_t = V$ の時，上述のラベルはグラフ中の任意の最短経路をカバーする．各ノードのラベルサイズは，グラフの直径 D とグラフに対して定義される Highway Dimension h という値を用いて，$\mathcal{O}(h \log D)$ にできることが知られている [18]．道路ネットワークでは Highway Dimension h は小さい値になると考えられている [18]．しかし，このようなラベルを算出するために用いる最小化問題が一般には NP 困難であることが知られており [19]，ラベル算出にはヒューリスティクスが用いられる．Hub Labeling [15] では Contraction Hierarchy [13] を活用して，ラベルサイズを小さくすることに成功している．

2.3 Pruned Highway Labeling

Pruned Highway Labeling (PHL)[16] はラベリング法を用いた経路探索クエリアルゴリズムの一つであり，効率的なラベル生成アルゴリズムが特徴である．しかし，無向グラフにしか適用できないという欠点がある．本論文のラベル生成アルゴリズムにおいても PHL のアイデアを使用するため，PHL の概要を述べる．

まず，PHL においては，グラフのノードを複数のノード集合に分解する．その時，各ノード集合は最短経路となっているようにする．すなわち，無向グラフ $G = (V, E)$ が k 個の最短経路 $\tilde{P}_1 = (\tilde{V}_1, \tilde{E}_1), \cdots, \tilde{P}_k = (\tilde{V}_k, \tilde{E}_k)$ に分解される時，$V = \tilde{V}_1 \cup \cdots \cup \tilde{V}_k$，任意の i, j $(i \neq j)$ に対して $\tilde{V}_i \cap \tilde{V}_j = \emptyset$ である．

V 中の各ノード v は $\{\tilde{V}_1, \tilde{V}_2, \cdots, \tilde{V}_k\}$ の部分集合 $\Pi(v) \subset \{\tilde{V}_1, \tilde{V}_2, \cdots, \tilde{V}_k\}$ を以下の性質を満たすように保持することにする：任意の $s, t \in V$ に対し，あるノード集合 $\tilde{V}_i \in \Pi(s) \cap \Pi(t)$ が存在し，$\tilde{V}_i$ が s-t 最短経路をカバーする．直感的には，2.2 節のラベリング法ではグラフ中のすべての最短経路に対して少なくとも一つの通過ノードが含まれるようにラベルを生成したが，PHL ではすべての最短経路に対して少なくとも一つの通過パスが見つかるようにラベルを生成する．

この時，s-t 最短経路コストは以下のように算出できる．

$$d(s,t) = \min_{\tilde{V} \in \Pi(s) \cap \Pi(t)} \left\{ \min_{v,w \in \tilde{V}} \{d(s,v) + d(v,w) + d(w,t)\} \right\}$$

このうち，$d(v,w)$ については，$\tilde{V}$ が最短経路であることより，$\tilde{V}$ 中の各ノードが最短経路の端点 z までのコストを覚えておくことで，$d(v,w) = |d(w,z) - d(v,z)|$ と計算できる．上記の最短経路コスト計算式ではノード s から $\tilde{V}$ 中の全ノードへの最短経路コストが必要になるが，実際には全ノードへのコストを保持する必要はない．ノード s から $v \in \tilde{V}$ への最短経路（の一つ）が $x \in \tilde{V}$ を通過する時，$d(s,v) + d(v,w) = d(s,x) + d(x,w)$ となり，上記の最短経路コスト計算式において同一の計算を複数回繰り返すことになるので，このような v についてはコストを保持する必要がない．ノード s から $v \in \tilde{V}$ へのどの最短経路も $\tilde{V}$ 中の他のノードを通過しない時，v をジャンクションと呼ぶことにする．この時，各ノード s は $\tilde{V}$ 中の各ジャンクションまでのコストを保持すればよい．実際の道路ネットワークにおいては，各最短経路に対するジャンクションの数は大きくならないので，ラベルのサイズを小さくすることができる．

次に，各ノード v に対する $\Pi(v)$ の算出方法について説明する．$\tilde{V}_i$ の各ノードからノード v への最短経路について，$\tilde{V}_1, \cdots, \tilde{V}_{i-1}$ によってカバーされない最短経路が存在する時，$\tilde{V}_i$ を $\Pi(v)$ に含める．$\tilde{V}_1$ については，すべてのノードの $\Pi(v)$ に含まれる．$V = \tilde{V}_1 \cup \cdots \cup \tilde{V}_k$ であることから，グラフ中のすべての最短経路がカバーされることが示される．上記は $\tilde{V}_1$ から $\tilde{V}_k$ の順に計算していくことで求めることができるため，効率的なアルゴリズムを設計できる．

また，$\{\tilde{V_1}, \tilde{V_2}, \cdots, \tilde{V_k}\}$ を高速道路などの重要な最短経路が前にくるように並べることで，より多くの経路が前半でカバーされることになり，ラベルサイズが小さくなる．

3. 提案アルゴリズム

PHL は無向グラフでしか使用できないという欠点がある．有向グラフではノード s から t への最短経路とノード t から s への最短経路は一般には異なるので，ある最短経路を構成するノードの集合から誘導部分グラフを作成した際，最短経路を逆向きに辿る経路ついては一般には最短経路になっていない（そもそも逆向きの辺が存在するとは限らない）．よって，PHL のようにグラフを複数のノード集合に分割することはできない．

本論文では有向グラフで使用可能なアルゴリズムについて検討する．まず提案アルゴリズムの概要について 3.1 節で説明する．ラベル生成時 PHL と同様の枝刈りダイクストラ法を行うと，前処理データのサイズが大きくなってしまう．そこでラベルサイズを小さくするようなラベル生成法について 3.3 節で検討する．

3.1 提案アルゴリズム概要

まず，グラフの最短経路分解を定義する．

定義 3.1 (グラフの最短経路分解) 有向グラフ $G = (V, E)$ とその部分グラフの集合 $\left\{\tilde{P_1} = (\tilde{V_1}, \tilde{E_1}), \cdots, \tilde{P_k} = (\tilde{V_k}, \tilde{E_k})\right\}$ が与えられた時，すべての部分グラフが最短経路であり，かつ以下を満たす時，部分グラフの集合をグラフの最短経路分解と呼ぶ．

$$E = \tilde{E_1} \cup \cdots \cup \tilde{E_k}$$

かつ

$$\text{任意の } i, j \ (i \neq j) \text{ に対して} \tilde{E_i} \cap \tilde{E_j} = \emptyset$$

このような最短経路分解を求めるアルゴリズムについては 3.2 節で説明する．

次に，辺のカバーと辺集合のカバーについて定義する．

定義 3.2 (辺のカバー) 有向グラフ $G = (V, E)$ と s-t 最短経路により構成される部分グラフ $P_{s,t} = (V_{s,t}, E_{s,t})$ が与えられた時，辺 $e \in E$ が $e \in E_{s,t}$ であるならば，辺 e は s-t 最短経路をカバーするという．

定義 3.3 (辺集合のカバー) 有向グラフ $G = (V, E)$ と s-t 最短経路により構成される部分グラフ $P_{s,t} = (V_{s,t}, E_{s,t})$, 辺集合 E_0 が与えられた時, ある辺 $e \in E_0$ が存在し, $e \in E_{s,t}$ であるならば，辺集合 E_0 は s-t 最短経路をカバーするという．

有向グラフに対応するためにカバープロパティについて以下のように変更する．

定義 3.4 (提案手法のカバープロパティ)
有向グラフ $G = (V, E)$ と最短経路分解 $\left\{\tilde{P_1} = (\tilde{V_1}, \tilde{E_1}), \cdots, \tilde{P_k} = (\tilde{V_k}, \tilde{E_k})\right\}$，各ノード $v \in V$ に対する最短経路の集合 $\Pi_0(v) \subset \{\tilde{P_1}, \tilde{P_2}, \cdots, \tilde{P_k}\}$, $\Pi_1(v) \subset \{\tilde{P_1}, \tilde{P_2}, \cdots, \tilde{P_k}\}$ が与えられた時，下記を満たすならば，$\Pi_0(v), \Pi_1(v)$ はカバープロパティを満たすという．任意の異なる 2 ノード $s, t \in V$ に対し，ある辺集合 $\tilde{P_i} \in \Pi_0(s) \cap \Pi_1(t)$ が存在し，$\tilde{E_i}$ が s-t 最短経路をカバーする．

出発地ノードと目的地ノードが同一の場合最短経路は辺を含まないため，辺でカバーすることができないことに注意する．クエリにおいて出発地と目的地が同一ノードの場合は前処理で作成したデータを使わずに，即座にコスト 0 を返すことにする．ここで，$\Pi_0(v)$ は出発地となるノード用のものであり，$\Pi_1(v)$ は目的地となるノード用のものである．PHL と同様にノード集合をベースとしたカバープロパティを定義しようとすると，$\tilde{P_i}$ を逆向きに通過するような最短経路に対する対処が困難になるため今回は採用しない．

この時，s-t 最短経路コストは以下のように算出できる．

$$d(s,t) = \min_{\tilde{P} \in \Pi_0(s) \cap \Pi_1(t)} \left\{ \min_{v,w \in \tilde{V}} \{d(s,v) + d_{\tilde{P}}(v,w) + d(w,t)\} \right\}$$

$d_{\tilde{P}}(v, w)$ は $\tilde{P}$ 上のノード v からノード w までの最短経路コストであるが，$\tilde{P}$ で構成される最短経路においてノード v の通過順がノード w よりも前である時以外は無限大を返す．

このうち，$d_{\tilde{P}}(v, w)$ については，$\tilde{P}$ が最短経路であることより，$\tilde{P}$ 中の各ノードが最短経路の終点ノード z までのコストを覚えておくことで計算できる．ノード v の通過順がノード w よりも前である場合，$d_{\tilde{P}}(v, w) = d(v, z) - d(w, z)$ となる．上記の最短経路コスト計算式では，各ノードから $\tilde{P}$ の全ノードへの最短経路コストが必要になるが，実際には全ノードへのコストを保持する必要はない．これについては 3.3 節で述べる．PHL においては，ジャンクションへの最短経路コストのみを保持することでラベルサイズを小さくすることができたが，有向グラフにおいては，ジャンクションの数は小さくならない．例えば，$\tilde{E}$ で構成される最短経路の終点ノード付近のノードについては，$\tilde{E}$ のほぼすべての辺がジャンクションとなることが考えられる．なぜならば，ノードから $\tilde{E}$ 中の辺までの最短経路が他の $\tilde{E}$ 中の辺を通過することがないからである（辺は有向辺であることに注意する）．よって，この PHL と同様の手法を用いるとラベルサイズが膨大になってしまう．それに伴い前処理計算時間も大きくなる．また，クエリの計算時間はラベルサイズに対する線形時間であるため，クエリも低速である．よって，3.3 節で述べる新たなアルゴリズムが必要である．

Algorithm 2 辺ベースのグラフ分解

```
1: procedure DECOMPOSE(G)                                   ▷ G = (V, E)
2:     S[e] ← 1 for all e ∈ E                               ▷ 辺状態
3:     while S[e] が 1 である e が存在 do
4:         P を空のパスとする
5:         最短経路の出発辺 r をランダムに決定する.
6:         P ← r を出発地とする最短経路のうち全ての辺 e で
   S[e] = 1 であるもの
7:         S[f] ← 0 for all f ∈ E_P          ▷ E_P：パス P の辺集合
8:         P を出力に追加.
9:     end while
10: end procedure
```

3.2 グラフの最短経路分解

Algorithm 2 に，辺ベースのグラフ分解を行うアルゴリズムを示す．最短経路を一つ生成したのち，その最短経路に含まれない辺の中から再び最短経路を生成することを繰り返すことで入力グラフを複数の最短経路に分解する．最短経路算出については，まず経路始点となる辺を選択したのち，その辺を出発地とする最短経路を算出する．5 行目の出発辺決定処理ではより多くの最短経路をカバーする辺を選ぶことにより性能が向上することが考えられる．そのためのヒューリスティクスを 4.1 節で説明する．

3.3 前処理データの算出

前節で作成されたグラフの最短経路分解を活用して，各ノードのラベルを算出する処理について述べる．有向グラフ $G = (V, E)$ とその最短経路分解 $\left\{\tilde{P}_1 = (\tilde{V}_1, \tilde{E}_1), \cdots, \tilde{P}_k = (\tilde{V}_k, \tilde{E}_k)\right\}$ が与えられた際のラベル生成処理について説明する．全体の流れとしては，PHL と同様に $\tilde{P}_1$ から $\tilde{P}_k$ の順に処理を行う．$\tilde{P}_i$ の各ノードから全ノード $v \in V$ への最短経路について，$\tilde{P}_1, \cdots, \tilde{P_{i-1}}$ によってカバーされない最短経路が存在する時，$\tilde{P}_i$ を v のラベルに含める．これを計算するために PHL と同様に枝刈りダイクストラ法を用いる．$\tilde{P}_1$ については，グラフ中の全ノードのラベルに含まれる．

次に前処理データの構造について述べる．各ノードは三つ組：(分解された最短経路の番号，最短経路上のノードまでのコスト，最短経路上のノードから最短経路終点までのコスト) を要素とする配列を保持する．3.1 節で説明した通り，$\tilde{P}_1 \in \Pi_0(v)$ の時 $\tilde{P}_1$ 上のすべてのノードに対する三つ組を保持すれば正しく経路探索クエリを計算できるが，もっとラベルサイズを小さくすることができる．

ラベルサイズを小さくするための基本的なアイデアは $\tilde{P}_i$ 中で v を目的地（もしくは）出発地とする最短経路において絶対に使用されない辺があるということである．

定理 3.5 (最短経路をカバーしない辺) ある辺 $e = (u, v)$ とノード t を目的地とする最短経路を考える．この時下記を満たすならば，辺 e はノード t を目的地とするどの最短経路もカバーしない．

Algorithm 3 前処理データの生成

```
1: procedure PREPROCESS(G)                                  ▷ グラフ G
2:     {P̃_1, ··· , P̃_k} ← DECOMPOSE(G)                     ▷ Algorithm 2
                                                  ▷ P̃_i = (Ṽ_i, Ẽ_i)
3:     L_0(v) ← ∅ for all v ∈ G
4:     L_1(v) ← ∅ for all v ∈ G
5:     for i ← 1 to k do
6:         L_1 ← UPDATELABELSDEST(G, P̃_i, L_0, L_1)
                                                          ▷ Algorithm 4
7:         L_0 ← UPDATELABELSORIG(G, P̃_i, L_0, L_1)
8:     end for
9:     return L_0, L_1
10: end procedure
```

$$d(u,t) < c(e) + d(v,t)$$

証明. 背理法で示す．ある出発地ノード s が存在し，$e = (u, v)$ が s-t 最短経路をカバーすると仮定する．この時ノード u, v を通過する s-t 最短経路が存在し，最短経路コストは $d(s,u) + c(e) + d(v,t)$ である．しかし，仮定より $d(s,u) + d(u,t) < d(s,u) + c(e) + d(v,t)$ となり，これはよりコストの小さい s-t 経路が存在することを意味する．これは e を通過する経路が s-t 最短経路であることに矛盾する．よって背理法の仮定は矛盾であり，辺 e はノード t を目的地とするどの最短経路もカバーしない． □

出発地ノードについても上記と同様の定理が成り立つ．これを活用してアルゴリズムを設計する．最短経路分解 $\left\{\tilde{P}_1, \cdots, \tilde{P}_k\right\}$ に対し，$\tilde{P}_1$ から $\tilde{P}_k$ の順に処理を行い，$\tilde{P}_i$ の処理では $\tilde{P}_1$ から $\tilde{P_{i-1}}$ で作成されたラベルでカバーされない経路のためのアイテムをラベルに追加する．以下では，ノード v の出発地側ラベルを $L_0[v]$，目的地側ラベルを $L_1[v]$ と表し，$\mathcal{L}_0 = (L_0[1], \cdots, L_0[|V|])$, $\mathcal{L}_1 = (L_1[1], \cdots, L_1[|V|])$ とする．ラベル生成アルゴリズムを Algorithm 3 に示す．この中で，出発地用ラベルの更新関数 UPDATELABELSORIG と目的地用ラベルの更新関数 UPDATELABELSDEST を呼び出しているが，このうち関数 UPDATELABELSDEST の処理を Algorithm 4 に示す．関数 UPDATELABELSORIG については関数 UPDATELABELSDEST と同様の処理となる．Algorithm 4 の 4 行目について，実際には毎回空のキューを生成する必要はなく，少し工夫することで前のイテレーションのキューの状態を引き継ぐことができる．また，処理中に使用されている関数 QUERY については現在のラベル情報から得られる最短経路コストを算出する関数であり，擬似コードが Algorithm 5 に示されている．前処理途中では最短経路コストを返すとは限らないが，前処理終了後は常に最短経路コストを返す．

定理 3.6 (Algorithm 3 の正当性) Algorithm 3 により得られたラベルを用いた経路探索クエリは常に最短経路コストを返す．

証明. Algorithm 4 において，$\tilde{P}_i$ がカバー可能で $\tilde{P}_1$ か

Algorithm 4 目的地側ラベルのアップデート

```
1: procedure UpdateLabelsDest(G, P̃_i, L_0, L_1)
     ▷ グラフ G, 最短経路 P̃_i, 出発地用ラベル L_0, 目的地用ラベル L_1
                                                  ▷ P̃_i = (Ṽ_i, Ẽ_i)
2:     n ← |Ṽ_i|                                  ▷ Ṽ_i のサイズを n とする
                                  ▷ Ṽ_i の経路始点を v_1, 終点を v_n とする
3:     for j ← n − 1 to 1 do
4:         空のキュー Q を生成する
5:         Q.push(d(v_j, v_{j+1}), v_{j+1}, j + 1)
6:         Q.push(0, v_j, j)
7:         while Q のアイテムに第 3 項の値が j + 1 のものが存在 do
8:             (d_v, v, ℓ) ← Q.pop()
9:             if ℓ = j + 1 then
10:                L_1(v) ← L_1(v) ∪ (i, d_v − d(v_j, v_{j+1}), d(v_{j+1}, v_n))
11:            end if
12:            for all v に隣接するノード w do
13:                d_w ← d_v + d(v, w)
14:                if d_w < Query(v_j, w, L_0, L_1) then
15:                    Q.push(d_w, w, ℓ)
16:                end if
17:            end for
18:        end while
19:    end for
20:    各ノードについて追加したラベルアイテムをソートする
21:    return L_1
22: end procedure
```

ら $P_{i-1}^{\sim}$ でカバーされない全ての最短経路に対するラベルが生成されることを示せばよい．まず，Algorithm 4 において 1 回もキューにプッシュされないノード x を考える．v_j-x 最短経路を考えると，最短経路上にある辺 $e = (\alpha, \beta)$ が存在し，以下を満たす：ノード α がキューにプッシュされた，かつノード β がキューにプッシュされなかった．この時，v_j-β 最短経路は $\tilde{E}_1$ から $E_{i-1}^{\sim}$ のいずれかでカバーされている．v_j-x 最短経路は v_j-β 最短経路と β-x 最短経路をつなぎ合わせたものであるため，$\tilde{E}_1$ から $E_{i-1}^{\sim}$ のいずれかでカバーされる．次に，キューにプッシュされたにもかかわらず，10 行目のラベル追加処理が行われない場合を考えると，これはキューアイテムの第 3 項が j の時である．これは定理 3.5 の式を満たす場合であり，対象の辺は最短経路をカバーしない．上記の 2 ケース以外では，ラベル追加処理が行われている．以上により示された． □

3.4 クエリ計算アルゴリズム

ノード s からノード t へ経路探索クエリは，ラベル $L_0(s), L_1(t)$ を参照することで計算できる．そのアルゴリズムを Algorithm 5 に示す．クエリは $L_0(s)$ と $L_1(t)$ をそれぞれ 1 回ずつスキャンする計算量で計算できるため，クエリの計算時間は $\mathcal{O}(|L_0(s)| + |L_1(t)|)$ となる．

Algorithm 5 提案手法の最短経路クエリ

```
1: procedure Query(s, t, L_0(s), L_1(t))
2:     cost ← ∞
3:     i, j ← 0
4:     while i < |L_0(s)| かつ j < |L_1(t)| do
5:         (p_0, a_0, d_0) ← L_0(s)[i]
                ▷ L_0(s)[i] は L_0(s) の i + 1 番目のアイテム．
              ▷ p_0 はパス ID．a_0 = d(s, v_0), d_0 = d(v_0, v_n)
                ▷ v_0 は p_0 上の接続ノード，v_n は p_0 の終点
6:         (p_1, a_1, d_1) ← L_1(t)[j]
                ▷ L_1(t)[j] は L_1(t) の j + 1 番目のアイテム
7:         if p_0 = p_1 then                    ▷ p_0 ∈ Π_0(s) ∩ Π_1(t)
8:             if a_0 > a_1 then
9:                 cost ← min{cost, d_0 + d_1 + a_0 − a_1}
10:                i ← i + 1
11:            else
12:                j ← j + 1
13:            end if
14:        else if p_0 < p_1 then
15:            i ← i + 1
16:        else if p_0 > p_1 then
17:            j ← j + 1
18:        end if
19:    end while
20:    return cost
21: end procedure
```

4. ヒューリスティクス

前処理データを小さくするための幾つかのヒューリスティクスについて述べる．

4.1 辺のレベル付け

3.2 節で述べた通り，グラフの最短経路分解 $\{\tilde{P}_1 \cdots, \tilde{P}_k\}$ の仕方により性能が大きく異なる．小さいインデックスの最短経路に主要な道路を割り当てると性能が向上する．そのために，グラフ中の辺を走行速度（距離コスト/時間コスト）をベースに 4 つのレベルに分解し，レベル順に最短経路を生成することにする．

4.2 ショートカットリンク

グラフの最短経路分解において，小さいインデックスの最短経路は多くのノードのラベルに含まれるためノード数が小さい方が生成されるラベルサイズが小さくなる．しかし，一方で小さいインデックスの最短経路はより多くの最短経路をカバーする必要がある．そこで小さいインデックスの最短経路において重要度が低い頂点を最短経路から削除し，残ったノード間をショットカット辺を追加することで接続することにする．

5. 実験

9th DIMACS Implementation Challenge [20] により提供されている米国の道路ネットワークを用いて，提案アル

表 1　性能比較結果（ニューヨーク）

手法	前処理時間 (sec)	サイズ (MB)	クエリ応答時間 (μs)
PHL	9.8	123.8	0.69
Ours(PTL)	226.7	867.9	1.47

ゴリズムの基礎的な評価を行った．使用したマシンの仕様は CPU: Intel Xeon E5-1620 3.70GHz，Memory: 256GB，OS: Linux CentOS 6.4 である．実装は C++11 を用いて行い，g++ 4.9.2 を用いてビルドを行った．並列処理は行わない．クエリー応答時間はランダムに選択した出発地，目的地での 100 回の平均値である．

5.1 Pruned Highway Labeling との性能比較

Github において公開されている PHL のソース[*1]を用いて，提案手法と PHL との性能比較を行った．比較には米国ニューヨークの道路ネットワーク（$|V| = 264K, |E| = 734K$）を用いた．

表 1 に PHL との性能比較結果を示す．ただし PHL は無向グラフに対して処理を行なったものであり，提案手法は有向グラフに対して処理を行なったものである．提案手法のデータサイズは出発地側ラベルと目的地側ラベルの合計である．無向グラフでは辺の情報量が有向グラフより少ない上に出発地側ラベルと目的地側ラベルと別々に保持する必要がないことに注意が必要である．提案手法の前処理データサイズは Pruned Highway Labeling の 7 倍であり，クエリ応答時間は 2.1 倍である．また前処理時間は 23 倍である．

まず，前処理データサイズの違いについては，提案手法のヒューリスティクスの性能が良くないことが考えられる．クエリ応答時間の違いについては，提案手法の前処理データサイズが大きいこととコード最適化が不十分であることが考えられる．前処理時間については，Algorithm 4 が遅いことが考えられる．

6. まとめ

道路ネットワーク上の最短経路クエリについて，新しいラベル生成アルゴリズムを提案し，その正当性の証明を行なった．提案アルゴリズムでは，効率的な前処理により有向グラフに対して効率的なラベルを生成することができる．今後は，様々なヒューリスティクスの実装および大規模グラフでの性能評価を行なっていく予定である．特に前処理時間を短くするためのヒューリスティクスや前処理データサイズを小さくするためのヒューリスティクスの開発を計画している．

[*1] https://github.com/kawatea/pruned-highway-labeling

参考文献

[1] Bast, H., Delling, D., Goldberg, A. V., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D. and Werneck, R. F.: Route Planning in Transportation Networks, *CoRR*, Vol. abs/1504.05140 (online), available from ⟨http://arxiv.org/abs/1504.05140⟩ (2015).

[2] Hart, P. E., Nilsson, N. J. and Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, pp. 100–107 (online), DOI: 10.1109/TSSC.1968.300136 (1968).

[3] Ikeda, T., Hsu, M.-Y., Imai, H., Nishimura, S., Shimoura, H., Hashimoto, T., Tenmoku, K. and Mitoh, K.: A fast algorithm for finding better routes by AI search techniques, *Vehicle Navigation and Information Systems Conference, 1994. Proceedings., 1994*, pp. 291–296 (online), DOI: 10.1109/VNIS.1994.396824 (1994).

[4] Caldwell, T.: On Finding Minimum Routes in a Network with Turn Penalties, *Commun. ACM*, Vol. 4, No. 2, pp. 107–108 (online), DOI: 10.1145/366105.366184 (1961).

[5] Winter, S.: Modeling Costs of Turns in Route Planning, *GeoInformatica*, Vol. 6, No. 4, pp. 345–361 (online), DOI: 10.1023/A:1020853410145 (2002).

[6] Orda, A. and Rom, R.: Shortest-path and Minimum-delay Algorithms in Networks with Time-dependent Edge-length, *J. ACM*, Vol. 37, No. 3, pp. 607–625 (online), DOI: 10.1145/79147.214078 (1990).

[7] BELLMAN, R.: ON A ROUTING PROBLEM, *Quarterly of Applied Mathematics*, Vol. 16, No. 1, pp. 87–90 (online), available from ⟨http://www.jstor.org/stable/43634538⟩ (1958).

[8] Ford, L. and Fulkerson, D.: *Flows in Networks*, Rand Corporation research study, University Press (1962).

[9] Meyer, U. and Sanders, P.: Delta-stepping: a parallelizable shortest path algorithm, Vol. 49, pp. 114–152 (2003).

[10] Chakaravarthy, V. T., Checconi, F., Murali, P., Petrini, F. and Sabharwal, Y.: Scalable Single Source Shortest Path Algorithms for Massively Parallel Systems, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 7, pp. 2031–2045 (online), DOI: 10.1109/TPDS.2016.2634535 (2017).

[11] Goldberg, A. V. and Harrelson, C.: Computing the Shortest Path: A Search Meets Graph Theory, *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 156–165 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1070432.1070455⟩ (2005).

[12] Gutman, R.: Reach-based routing: A new approach to shortest path algorithms optimized for road networks, pp. 100–111 (2004).

[13] Geisberger, R., Sanders, P., Schultes, D. and Delling, D.: Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks, *Proceedings of the 7th International Conference on Experimental Algorithms*, WEA'08, Berlin, Heidelberg, Springer-Verlag, pp. 319–333 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1788888.1788912⟩ (2008).

[14] Arz, J., Luxen, D. and Sanders, P.: Transit Node Routing Reconsidered, *Experimental Algorithms* (Bonifaci, V., Demetrescu, C. and Marchetti-Spaccamela, A., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 55–

66 (2013).

[15] Abraham, I., Delling, D., Goldberg, A. V. and Werneck, R. F.: A Hub-based Labeling Algorithm for Shortest Paths in Road Networks, *Proceedings of the 10th International Conference on Experimental Algorithms*, SEA'11, Berlin, Heidelberg, Springer-Verlag, pp. 230–241 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2008623.2008645⟩ (2011).

[16] Akiba, T., Iwata, Y., ichi Kawarabayashi, K. and Kawata, Y.: Fast Shortest-path Distance Queries on Road Networks by Pruned Highway Labeling, *Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pp. 147–154 (online), DOI: 10.1137/1.9781611973198.14 (2014).

[17] Akiba, T., Iwata, Y. and Yoshida, Y.: Fast Exact Shortest-path Distance Queries on Large Networks by Pruned Landmark Labeling, *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, New York, NY, USA, ACM, pp. 349–360 (online), DOI: 10.1145/2463676.2465315 (2013).

[18] Abraham, I., Fiat, A., Goldberg, A. V. and Werneck, R. F.: Highway Dimension, Shortest Paths, and Provably Efficient Algorithms, *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 782–793 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1873601.1873665⟩ (2010).

[19] Babenko, M. A., Goldberg, A. V., Kaplan, H., Savchenko, R. and Weller, M.: On the Complexity of Hub Labeling, *CoRR*, Vol. abs/1501.02492 (online), available from ⟨http://arxiv.org/abs/1501.02492⟩ (2015).

[20] Demetrescu, C., Goldberg, A. and Johnson, D.: *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, DIMACS series in discrete mathematics and theoretical computer science, American Mathematical Society (2009).