Vol.2016-AL-159 No.7
2016/9/23

道路ネットワーク上の経路探索クエリのための枝刈りツリーラベリングアルゴリズム

小池　敦[1,a)]　定兼　邦彦[2,b)]

概要：本論文では，道路ネットワーク上の経路探索クエリについて，新しいアルゴリズムを提案する．提案手法は Hub Labeling や Pruned Highway Labeling と同様，ラベリング法に基づくものである．前処理において，入力グラフを複数の木に分解し，各ノードが hub ノード集合の代わりに木の ID の集合を保持することでラベルサイズを削減させる．本論文ではアルゴリズムを実装し，米国の道路ネットワークを用いて評価を行うことで，提案アルゴリズムの有効性を示す．

キーワード：経路探索クエリ，道路ネットワーク，グラフアルゴリズム

Pruned Tree Labeling Algorithms for Shortest Path Queries in Road Networks

Atsushi Koike[1,a)]　Kunihiko Sadakane[2,b)]

Abstract: We propose a new algorithm for shortest path queries in road networks. The algorithm utilizes a labeling method in common with Hub Labeling and Pruned Highway Labeling. At the preprocessing phase, an input graph is decomposed into trees and each node in the graph stores some tree IDs instead of node IDs in Hub Labeling, which reduces the size of precomputed data. In this paper, we implement the algorithm and evaluate it using a US road network.

Keywords: Shortest path queries, road networks, graph algorithms

1. はじめに

本論文では道路ネットワーク上の最短経路問題を扱う．これは，重み付きグラフ及びグラフ上の出発地ノード，目的地ノードが与えられた時，出発地ノードから目的地ノードへのコスト最小のパスを求める問題である．コスト最小のパスは最短経路と呼ばれる．最短経路算出はカーナビや地理情報システムにおいて基本的な機能であり，これを高速化することは重要である．最短経路問題に対する古典的なアルゴリズムとしてはダイクストラ法や双方向ダイクストラ法が知られており，その改良として A*アルゴリズム [1] や双方向 A*アルゴリズム [2] がある．その後も多くの改良が行われており，主な先行研究は [3] にまとめられている．

最短経路算出クエリに高速に応答するための手法として以下で説明する 2 フェーズアプローチがあり，本論文でもこのアプローチを採用する．本アプローチは前処理フェーズとクエリフェーズからなる．前処理フェーズでは，グラフに対して前処理を行い，クエリ応答を高速化するための前処理データを作成する．クエリフェーズでは，前処理データを活用して，本フェーズで与えられた出発地，目的地に対する最短経路を計算する．2 フェーズアプローチは多くの地理情報システムで採用されており，多くの先行研究がある ([3] 参照)．

現在，クエリ応答時間最小の手法は *Hub Labeling* (HL)[4]

1　東北大学大学院情報科学研究科応用情報科学専攻
2　東京大学大学院情報理工学系研究科数理情報学専攻
a)　atsushi.koike.d7@tohoku.ac.jp
b)　sada@mist.i.u-tokyo.ac.jp

である．本手法では，前処理データとして，各ノードがラベルと呼ばれるデータを保持する．ラベルは hub と呼ばれるノードとコスト値のペアの集合で構成される．*Pruned Highway Labeling* (PHL)[5] は Hub Labeling と同様のアプローチを取っているが，前処理計算時間が Hub Labeling よりも小さいことが特徴である．入力グラフを複数の最短経路に分解したのち，各ノードは，hub ノードの代わりに最短経路 ID の集合と幾つかのコストを保持する．

本研究では，PHL の前処理データサイズを小さくすることを目的とする．提案アルゴリズムでは，まず，強最短路木を定義した後，入力グラフを複数の強最短路木に分解する．そして，各ノードは，hub ノードの代わりに強最短路木 ID の集合と幾つかのコストを保持する．本論文では，まず，定式化を行ったのち，グラフを複数の強最短路木に分解するためのアルゴリズムを提案する．その後，前処理のためのアルゴリズムとクエリのためのアルゴリズムを提案し，最後に実験結果を示す．

2. 準備

2.1 定式化

本論文では，入力グラフとして，正重みの無向グラフを扱う．道路ネットワークの性質より，グラフの各ノードの次数は定数で抑えられるものとする．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: 正重み無向グラフ $G = (V, E)$

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．

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，ラベリング法について概要を説明する．以下では無向グラフ G の s から t への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．

無向グラフ G 中のノード s,t の最短経路（の一つ）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，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．よって，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，多くの場合，V_w 中の全ノードへのコストを保持する必要はなく，以下のカバープロパティを満たすように保持すれば十分である．V_s, V_t 中の各ノード v は v からノード集合 $H(v)$ 中の全ノードへのコストを保持するものとする．この時カバープロパティは以下のようになる：任意の $s \in V_s, t \in V_t$ に対し，あるノード $w \in H(s) \cap H(t)$ が存在し，w が $s-t$ 最短経路をカバーする．

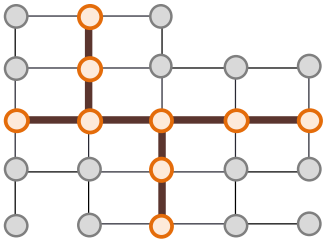
ノード v は $H(v)$ 中の全ノード w に対し，ペア $(w, d(v,w))$ を保持する．この集合は v のラベルと呼ばれ $L(v)$ と書くことにする．本論文では無向グラフを考えているので，$d(v,w) = d(w,v)$ であることに注意する（参考まで有向グラフを扱う時には $(w, d(w,v))$ と $(w, d(v,w))$ は別々に扱う必要がある）．ラベルを用いることで，$s \in V_s$ から $t \in V_t$ への最短経路コスト $d(s,t)$ は以下のように算出できる．

$$d(s,t) = \min \{d(s,w) + d(t,w) \mid w \in H(s) \cap H(t)\}$$

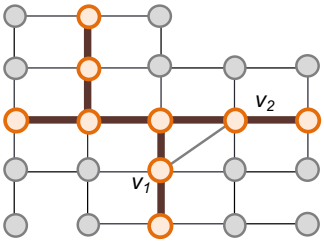
$V_s = V, V_t = V$ の時，上述のラベルはグラフ中の任意の最短経路をカバーする．各ノードの最適なラベルサイズは，グラフの直径 D，グラフに対して定義される Highway Dimension h を用いて，$\mathcal{O}(h \log D)$ となることが知られている [6]．しかし，ラベルサイズを最小化する問題は一般には NP 困難であることが知られており [7]，ラベル算出にはヒューリスティクスが用いられる．Hub Labeling [4] では Contraction Hierarchy [8] を活用して，ラベルサイズを小さくすることに成功している．

3. Pruned Tree Labeling

本章では，提案アルゴリズムについて述べる．まず，提案アルゴリズムにおいて用いられる基本的なデータ構造である強最短路木について説明したのち，アルゴリズムの概要について述べる．その後，前処理とクエリの詳細を述べる．



(a) クロージャプロパティを満たす
部分グラフ（茶色太線）



(b) クロージャプロパティを満たさない
部分グラフ（茶色太線）

図 1　クロージャプロパティの説明図．グラフのエッジの重みはすべて 1 とする．(a) の部分グラフ（茶色太線）は，部分グラフ上の任意のノード間の最短経路が部分グラフ上にあるため，クロージャプロパティを満たすが，(b) の部分グラフ（茶色太線）はノード v_1，v_2 間の最短経路が部分グラフ上にないため，クロージャプロパティを満たさない．(a) の部分グラフ（茶色太線）は強最短路木にもなっている．

3.1　強最短路木

まず，G の部分グラフに対して，クロージャプロパティを定義する．

定義 3.1 (クロージャプロパティ)　無向グラフ $G = (V, E)$ 及び，G の部分グラフ $H = (V_H, E_H)$ が与えられた時，V_H 中の任意のノード s, t に対して，$E_{s,t}$ 上のすべてのエッジが E_H に含まれるような $P_{s,t} = (V_{s,t}, E_{s,t})$ が存在するならば，H はクロージャプロパティを満たすと言う．

図 1 に例をあげる．実際の道路ネットワークにおいては，高速道路ネットワークの大部分がクロージャプロパティを満たしていると思われる．高速道路がクロージャプロパティを満たしているということは，始点と終点がともに高速道路上であれば，始点から終点までの最短経路は高速道路のみを使用する経路になるということである．もし，クロージャプロパティを満たさないのであれば，一旦，一般道に降りて，再び高速道路に乗るような最短経路が存在することになり，これは経済的に非効率である．次に，強最短路木を定義する．

定義 3.2 (強最短路木)　無向グラフ G の部分グラフ T が，木であり，かつクロージャプロパティを満たす時，T を強最短路木と呼ぶ．

図 1(a) は強最短路木の例にもなっている．強最短路木は根の取り方によらず，最短路木になっている．図 2 に実

図 2　強最短路木の具体例．道路ネットワークは 9th DIMACS Implementation Challenge [9] により提供されているニューヨークのものである．赤太線が強最短路木であり，丸で示したノードは後述の Algorithm 1 における根である．

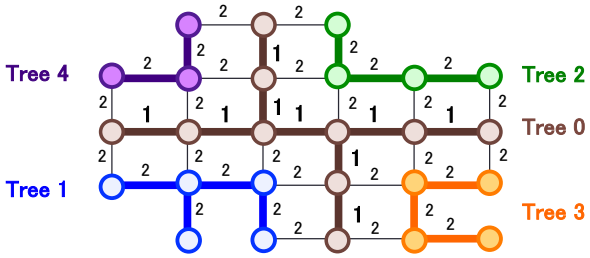


図 3　グラフの強最短路木への分解の例

際の道路ネットワーク上における強最短路木の例を示す．図 2 はニューヨークの道路ネットワークであり，赤太線が強最短路木になっている．丸で示したノードは後述の Algorithm 1 における根である．

次に，グラフの複数の強最短路木への分解について定義する．

定義 3.3 (グラフの強最短路木への分解)　無向グラフ $G = (V, E)$,
$\mathcal{T} = \{T_1 = (V_1, E_1), \cdots, T_k = (V_k, E_k)\}$
($T_1, \cdots, T_k$ はそれぞれ，G 中の強最短路木) が以下を満たすとき $\mathcal{T}$ を G の強最短路木への分解と呼ぶ．

$$V = V_1 \cup \cdots \cup V_k$$
$$V_i \cap V_j = \emptyset \text{ for all } i, j \text{ s.t. } 1 \leq i, j \leq k, i \neq j$$

強最短路木を用いたグラフの分解例を図 3 に示す．

筆者の知る限り，強最短路木分解の性質についてはほとんど知られていないが，早水ら [10] は完全グラフの場合に，グラフが一つの強最短路木に分解できるための必要十分条件を示している．

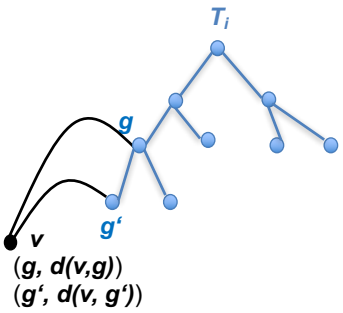


図 4　強最短路木に対するラベルの例

3.2　提案アルゴリズム概要

提案アルゴリズムでは，2.2 節で説明したラベリング法のアプローチを採用する．ただし，ラベルのデータ構造は 2.2 節とは異なる．

まず，入力グラフは複数の強最短路木 $\mathcal{T} = \{T_1 = (V_1, E_1), \cdots, T_k = (V_k, E_k)\}$ に分解される．強最短路木 T_1 は V_1 によりカバーされるすべての最短経路のためのラベルを生成する．ラベルのデータ構造については後述する．強最短路木 T_i $(2 \leq i \leq k)$ は，V_i によりカバーされる最短経路のうち，V_1〜V_{i-1} でカバーされないもののためのラベルを生成する．すなわち，$V_1 \cup \cdots \cup V_i$ によりカバーされるすべての最短経路のコストは T_1〜T_i のラベルにより算出できる．$V = V_1 \cup \cdots \cup V_k$ であるから，グラフ上のすべての最短経路のコストは T_1〜T_k のラベルにより算出できる．

次に，ラベルのデータ構造について述べる．ノード v が強最短路木 T_i に対して保持するラベルについて説明する．v から T_i の各ノード w への最短経路を考えると，T_i が強最短路木であることより，v から出発した経路が一旦 T_i のあるノード g に到達すると，その後は w まで T_i 上のノードを通過する（正確にはそのような最短経路が存在する）．このようなノード g を v の T_i に対する接点ノード (gateway node) と呼ぶこととする．v は T_i に対するラベルとして，接点ノードと接点ノードまでのコストのペア $(g, d(v, g))$ の集合を保持する（接点ノードは一つとは限らないことに注意する）．v の T_i に対するラベルの例を図 4 に示す．

次に，ラベルを用いた最短経路コスト算出方法について述べる．$s-t$ 最短経路コストは以下の式で算出できる．

$$d(s,t) = \min\{d(s,g_s) + \tilde{d}(g_s,g_t) + d(t,g_t) \mid (g_s, d(s,g_s)) \in L(s), (g_t, d(t,g_t)) \in L(t)\}$$

ただし，$\tilde{d}(g_s, g_t)$ は g_s と g_t が同一の木に属している時 $d(g_s, g_t)$，それ以外の時 ∞ であるとする．すなわち，$d(s,t)$ は，s から s の接点ノードまでのコストと t から t の接点ノードまでのコストと接点ノード間のコストを足したものとなっている．g_s, g_t が T_i に属しているとすると，接点ノード間のコスト $d(g_s, g_t)$ は以下のように算出できる．ま

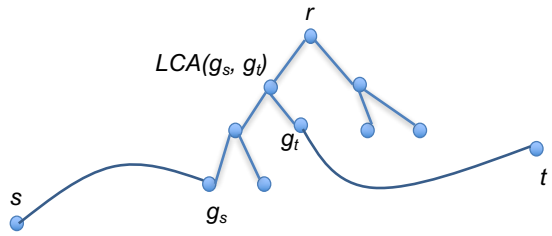


図 5　最短経路コスト計算の説明図

Algorithm 1 グラフの強最短路木への分解

```
1: procedure DECOMPOSE(G)                          ▷ G = (V, E)
2:     S[v] ← 1 for all v ∈ V                      ▷ ノード状態
3:     while S[v] が 1 である v が存在 do
4:         根 r を決定し，r を空の木 T に追加.
5:         while T にエッジを追加可能 do
6:             T 上の分岐ノード b を決定.
7:             P ← FINDCLOSEDPATH(G, T, b, S)
8:             パス P を T に追加.
9:         end while
10:        S[w] ← 0 for all w ∈ T
11:        T を出力に追加.
12:    end while
13: end procedure
```

ずあらかじめ T_i の根ノード r を決め，T_i の全ノードは根 r までのコストを保持しておく．すると，

$$\begin{aligned} d(g_s,g_t) &= d(g_s, LCA(g_s,g_t)) + d(g_t, LCA(g_s,g_t)) \\ &= d(g_s,r) + d(g_t,r) - 2 \cdot d(LCA(g_s,g_t), r) \end{aligned}$$

である．ここで，$LCA(g_s, g_t)$ は g_s と g_t の最小共通祖先であり，T_i に対して，LCA 索引を作成することにより定数時間で計算できる [11]．この状況の説明図を図 5 に示す．

なお，分解に使用する強最短路木をパスに限定すると，PHL と同様の前処理データ構造になることから，提案手法は PHL の自然な拡張であるとみなすことができる．

3.3　グラフの強最短路木への分解アルゴリズム

Algorithm 1 に，入力グラフを複数の強最短路木に分解するアルゴリズムを示す．強最短路木を一つ生成したのち，その木に含まれないノードの中から再び強最短路木を生成することを繰り返すことで入力グラフを複数の強最短路木に分解する．4 行目及び 6 行目に処理において，より多くの最短経路をカバーするノードを選ぶことが重要であるが，このために吉田により提案された Adaptive Coverage Centrality を近似計算する手法 [12] を活用する．

次に，Algorithm 1 の 7 行目の処理について詳細を述べる．Algorithm 1 の強最短路木算出処理は，クロージャプロパティを満たしながら，木に繰り返しパスを追加することで行われる．FINDCLOSEDPATH はそのようなパスを算出する関数であり，アルゴリズムを Algorithm 2 に示す．本アルゴリズムは T 上のノード b を始点とする最短経路

Algorithm 2 強最短路木に追加するパスの算出

```
1: procedure FindClosedPath(G, T, b, S)
   ▷ G = (V, E)，強最短路木 T = (V_T, E_T)，分岐ノード b ∈ V_T，
   ノード状態 S
2:     空のキュー Q を生成する
3:     d[v] ← ∞ for all v ∈ V
4:     Q.push(−d(b, w), w, φ, w, 1) for all w ∈ V_T
5:     while Q が空でない do
6:         (d_v, v, u, t, c) ← Q.pop()
7:         if d_v < d[v] then
8:             d[v] ← d_v
9:             prev[v] ← u
10:            orig[v] ← t
11:            candidate[v] ← c
12:            for all v に隣接するノード w do
13:                d_w ← d_v + c(v, w)
14:                if d_w < d[w] かつ w ∉ V_T then
15:                    Q.push(d_w, w, v, t, c ∧ S[w])
16:                end if
17:            end for
18:        end if
19:    end while
20:    orig[v] = b かつ candidate[v] = 1 である
       ノードのうち，d[v] 最大のノード v を求める．
21:    prev[v] をたどり b − v 最短経路 P を出力．
22: end procedure
```

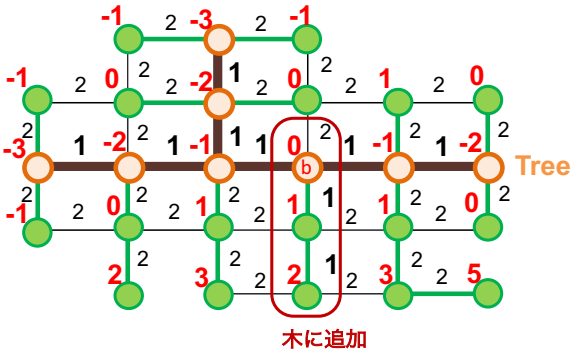


図 6　強最短路木へのパス追加の例．強最短路木（茶色）のノード b にパスを追加する．ノード左上のコストは Algorithm 2 により算出された $d[v]$ である．

のうち，T に追加可能なものを算出する．アルゴリズム中で，エッジ (v, w) の重みは $c(v, w)$ と表す．また，キューに挿入するアイテムは $(d, v, \text{prev}, \text{orig}, \text{candidate})$ のような 5 つ組であり，キーは d である．本アルゴリズムにおいて，一つのノードが複数回キューに挿入されることがあるが，その回数は道路ネットワークにおいては高々定数回である（ノードの次数は定数で抑えられるため）．Algorithm 2 により強最短路木にパスを追加する例を図 6 に示す．

次に強最短路木 T に Algorithm 2 により見つかったパス P を追加しても，強最短路木となることを証明する．なお，Algorithm 2 中の変数 S については，アルゴリズムの正当性に影響を及ぼさないので，以下では考えない．

定理 3.4 (Algorithm 2 の正当性)　無向グラフ G，G 中の強最短路木 $T = (V_T, E_T)$ 及び T 中の分岐ノード b が与えられた時，Algorithm 2 の出力がパス $P = (V_P, E_P)$ であったとする．この時，T に P を連結した部分グラフ T' は強最短路木である．

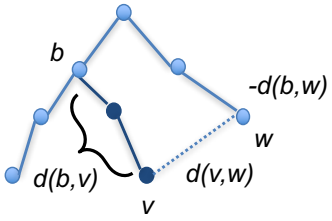


図 7　定理 3.4 の証明の説明図

証明.　T' が木であり，クロージャプロパティを満たすことを示す．そのためには，任意の $v \in V_P \setminus \{b\}$ が $v \notin V_T$ であること，及び，任意の $v \in V_P \setminus \{b\}$ と任意の $w \in V_T$ において，ある $v - w$ 最短経路 Q が存在し，Q が T' 上にあることを示せばよい．

まず，前者を示す．任意のノード $v \in V_P \setminus \{b\}$ において，先行ノードをたどると b に到達することから $d[v]$ は正の値になる．一方で，任意のノード $w \in V_T$ において，$d[w]$ は 0 以下の値となる（Algorithm 24 行目による）．よって，任意のノード $v \in V_P \setminus \{b\}$ は $v \notin V_T$ である．

次に後者を示す．任意の $v \in V_P \setminus \{b\}$ と任意の $w \in V_T$ において，$v - w$ 最短経路を考える．この状況の説明図を図 7 に示す．アルゴリズムより $d[w] = (-1) \cdot d(b, w)$ であり，$d[v] = d(b, v)$ である．今，v の先行ノードをたどると b に到達することから $d[v] \leq d[w] + d(v, w)$，すなわち，$d(b, v) \leq (-1) \cdot d(b, w) + d(v, w)$ である．よって，$d(v, w) \geq d(b, w) + d(b, v)$ が成立する．よって，T' 上を通過する v から w への経路は最短経路である．以上により示された．　□

3.4　前処理データの算出

前節で作成されたグラフの木への分解を活用して，各ノードのラベルを算出する処理について述べる．アルゴリズムは Pruned Highway Labeling [5] において提案されている枝刈ダイクストラ法と同様である．強最短路木 T_i $(1 \leq i \leq k)$ のための処理においては，V_i によりカバーされる最短経路のうち，生成済みのラベルでカバーされない経路のための要素をラベルに追加する．以下では，ノード v のラベルを $L[v]$ と表し，$\mathcal{L} = \{L[1], \cdots, L[|V|]\}$ とする．

3.5　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 $L(s), L(t)$ を参照することで計算できる．$L(s)$ の各要素と $L(t)$ の各要素を比較すれば良いので，ナイーブに行うと計算時間は $\mathcal{O}(|L(s)| * |L(t)|)$ となる．しかし，この計算時間

Algorithm 3 前処理データの生成

```
1: procedure PREPROCESS(G)                          ▷ グラフ G
2:     T ← DECOMPOSE(G)                             ▷ Algorithm 1
3:     for all T ∈ T do
4:         各ノードが根までのコストを保持する
5:         LCA 算出用の索引を作成する
6:     end for
7:     L(v) ← ∅ for all v ∈ G
8:     k ←  the size of T
9:     for i ← 1 to k do
10:        空のキュー Q を生成する
11:        Q.push(0, v, v) for all v ∈ T
12:        while Q が空でない do
13:            (d_v, v, t) ← Q.pop()
14:            if d_v < QUERY(v, t, L) then
15:                L(v) ← L(v) ∪ (t, d_v)
16:                for all v に隣接するノード w do
17:                    d_w ← d_v + c(v, w)
18:                    if d_w < d[w] then
19:                        Q.push(d_w, w, t)
20:                    end if
21:                end for
22:            end if
23:        end while
24:    end for
25:    L を出力する
26: end procedure
```

は $\mathcal{O}(|L(s)|+|L(t)|)$ に改善できる．ラベルの要素が木あたり一つしかない場合は，ラベルの要素を木 ID でソートしておくことで，計算時間を $\mathcal{O}(|L(s)|+|L(t)|)$ にできる．そこで，一つの木に対して，複数のラベル要素がある場合の処理の概要を説明する．まずラベルの要素を接点の深さ優先探索順にソートしておく．そして，深さ優先順にラベルをチェックする．その際，各接点間の LCA の位置にラベル要素がない場合は，ダミーのラベル要素を追加する．これにより，各ラベル要素に対し，深さ優先順の前後のラベル要素のみをチェックすれば良いようになる．この状況を図 8 に示す．赤ばつ印は s に対するラベル要素の接点ノード，青ばつ印は t に対するラベル要素の接点ノード，黒ばつ印はダミーのラベル要素である．この状況においては，各ラベル要素を深さ優先探索順に 1 回ずつチェックすれば良いので，計算時間は $\mathcal{O}(|L(s)|+|L(t)|)$ となる．

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 入力グラフからの低次数ノードの削除

Pruned Highway Labeling と同様に次数の低いノードについてはグラフから削除することにし，もし出発地，目的地として削除されたノードが指定された場合には，隣接ノードから求めることを考える．$s-t$ 最短経路コストは，s の隣接ノードを用いて以下のように書くことができる（t についても同様）．

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

ここで，$c(s,w)$ は辺 (s,w) のコストである．よって，s のすべての隣接ノード w に対して $c(s,w)$ を保持しておくことで，s をグラフから削除しても，$s-t$ 最短経路コストを算出することができる．なお，s,t ともにグラフから削除されている場合，クエリを (s の次数)×(t の次数) 回実行する必要がある．そのため，次数の大きいノードを削除すると，クエリ計算時間が大きくなる．次数 k 以下のノードを削除するとき，レベル k のグラフ縮約と呼ぶことにする．本論文では，レベル 2 のグラフ縮約を行う．

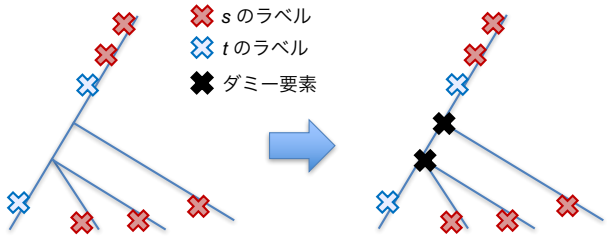


図 8 クエリの高速計算の説明図．左図において，赤ばつ印は s に対するラベル要素の接点ノード，青ばつ印は t に対するラベル要素の接点ノードである．これに，右図のように黒ばつ印のようなダミーのラベル要素を追加する．この時，各ラベル要素を深さ優先探索順に 1 回ずつチェックすれば良い．

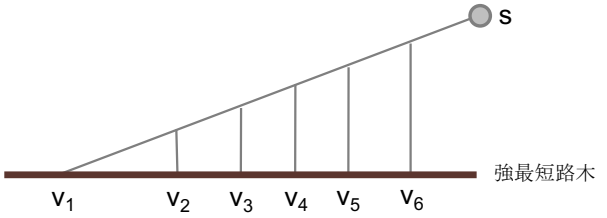


図 9 強最短路木からの非重要ノード削除が有効となるシチュエーション

4.2 強最短路木からの非重要ノードの削除

図 9 のような状況を考える．$v_1,\cdots,v_6$ は強最短路木中のノードであり，s からこの木に対するラベルを生成することを考える．この時 $v_1,\cdots,v_6$ は全て接点ノードになる．よって s はこれら全てのノードへのコストを保持する必要がある．しかし，$v_2,\cdots,v_5$ は s から自身のノードへの最短経路のみをカバーするため，他の経路探索には役立たない．ラベルサイズを小さくするためにより少ないノードで多くの最短経路をカバーしたいが，これらのノードはラベルサイズを大きくする要因となっている．そこで，生成した木の各ノードに対し重要度（カバーする最短経路数）をヒューリスティクスで見積もり，それが小さいノードは木から削除することにする．

表 1　性能比較結果（米国全土）

手法	前処理時間 (h:m)	サイズ (GB)	クエリ応答時間 (μs)
PHL	0:33	20.0	1.1
Ours(PTL)	76:13	7.8	149.9

表 2　縮約レベルによる性能の違い（米国コロラド州）

手法 (縮約レベル)	前処理時間 (m:s)	サイズ (MB)	クエリ応答時間 (μs)
PHL (Lv1)	0:09	143	0.7
PTL (Lv0)	9:12	131	6.7
PTL (Lv1)	7:11	97	8.3
PTL (Lv2)	3:25	56	21.5

5. 実験

9th DIMACS Implementation Challenge [9] 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，提案アルゴリズムの評価を行った．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．LCA 索引は簡潔データ構造ライブラリの一つである sdsl-lite[13] を用いて作成した．また，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．このソースではレベル 1 のグラフ縮約が行なわれている．比較には米国全土の地図（($|V| = 24M, |E| = 58M$)）を用いた．

表 1 に PHL との性能比較結果を示す．提案手法の前処理データサイズは，Pruned Highway Labeling の 39%になっていることがわかる．しかし，クエリ応答時間は 136 倍になっている．これは LCA を求める処理に時間がかかっていることやグラフ縮約による処理時間の増加が考えられる．また，前処理計算時間は 138 倍になっている．これは，前処理においてクエリを多用することから，クエリの計算時間が大きいことが主な要因と思われる．

5.2 グラフ縮約レベルの比較

4.1 節のグラフ縮約のレベルによる性能の違いを比較する．比較には米国コロラド州の地図（($|V| = 436K, |E| = 1057K$)）を用いた．また，提案手法に関して，5.1 節と比較し幾つかヒューリスティクスの変更を行っている．

表 2 に縮約レベルによる性能の違い及び PHL との比較結果を示す．まず，提案手法に関して，Lv1 縮約は縮約なしと比べ前処理データサイズが 74%になっていることがわかる．また，Lv2 の縮約は Lv1 と比べ前処理データサイズが 58%になっている．また，Lv2 の前処理時間は Lv1 前処理時間の 48%になっている．しかし，クエリ計算時間は 2.6 倍になっている．これは 4.1 節の説明したグラフ縮約によるクエリ計算時間の増加によるものと考えられる．また，Lv1 縮約の PTL を PHL と比較すると，PTL の前処理データサイズは PHL の 68%になっている．

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

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，新しいアルゴリズムを提案した．提案アルゴリズムでは，グラフの分解に木を用いることで Pruned Highway Labeling の前処理データサイズを小さくする．米国の道路ネットワークを用いて評価を行うことで，実際に前処理データサイズが小さくなることを確認した．

その中で，道路ネットワークのための性質としてクロージャプロパティを定義し，道路ネットワークをクロージャプロパティを満たす木（強最短路木）に分解するアルゴリズムを提案した．

今後は，ラベルサイズをさらに小さくすることについて検討していきたい．

参考文献

[1] 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).

[2] 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).

[3] 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).

[4] 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).

[5] 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).

[6] 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 Sympo-*

sium 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).

[7] 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).

[8] 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).

[9] 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).

[10] Hayamizu, M., Endo, H. and Fukumizu, K.: A Characterization of Minimum Spanning Tree-like Metric Spaces, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, Vol. PP, No. 99, pp. 1–1 (online), DOI: 10.1109/TCBB.2016.2550431 (2016).

[11] Arroyuelo, D., Cánovas, R., Navarro, G. and Sadakane, K.: Succinct Trees in Practice, *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, ALENEX '10, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 84–97 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2790231.2790240⟩ (2010).

[12] Yoshida, Y.: Almost Linear-time Algorithms for Adaptive Betweenness Centrality Using Hypergraph Sketches, *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, New York, NY, USA, ACM, pp. 1416–1425 (online), DOI: 10.1145/2623330.2623626 (2014).

[13] Gog, S., Beller, T., Moffat, A. and Petri, M.: From Theory to Practice: Plug and Play with Succinct Data Structures, *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pp. 326–337 (2014).