OR 事典 Wiki

最短路検索

宮本裕一郎，藤澤　克樹，久保　幹雄

与えられたネットワーク上での2地点間最短路検索を最短路検索とよぶ．ネットワークデータへの前処理を含む，多くの問い合わせに高速に応答するための，データ構造とアルゴリズムの工夫を紹介する．

1. 最短路検索とは

枝重み付きネットワーク上で始点 s から終点 t に至る経路のうち，その枝重み和が最小となる経路を st-最短路とよぶ（本稿では st-最短路長を $d(s,t)$ で表す）．最短路検索は st-最短路を高速に検索することを主な目的とする．その厳密な定義はないがおおむね以下の通りである．

> 入力データとして与えられた枝重み付きネットワーク $G=(V,E)$ に**前処理**を施して，付加データを生成する．始点 s，終点 t の問い合わせに対して入力データと付加データを使って st-最短路（あるいは $d(s,t)$）を**検索**する．

ネットワークデータは滅多に変化しないが st-最短路の問い合わせは頻繁にあるような，交通におけるナビゲーションなどを主な応用とする．

交通ネットワークなどを応用とする多くの最短路検索においては，入力の枝重みが非負であることが多い．本稿では枝重みが非負の最短路検索のみを扱う．

最短路検索手法として，最適解を出力するとは限らない，近似的な解を出力する手法も考えられる．本稿では最適解のみを出力する手法を扱う．また，計算幾何の手法を用いて連続領域における最短路を検索する手法の研究もあるが，本稿ではネットワークを入力データとする最短路検索のみを扱う．

例えば，前処理を全くせずに，始点・終点の問い合わせに対して**ダイクストラ（Dijkstra）法**を実行し最短路を出力するのも最短路検索手法の一つといえる．本稿で紹介する手法はいずれも，道路ネットワークなどが入力データとして与えられた場合には，検索速度はダイクストラ法に比べて大幅に速くなるが，最悪の場合の計算量は単純にダイクストラ法を実行する手法と変わらない．

2. 代表的な手法の効率の比較

最短路検索手法の性能に関する重要な要素は，検索時の応答の速さ，付加するデータの大きさ，前処理にかかる時間，の3つである．表1に，次節以降で紹介する代表的な手法のおおまかな比較を示す．入力データが全米道路ネットワーク（約2300万点，約5800万枝）程度の大きさであることを想定している．ここで「検索」はダイクストラ法で最短路を計算する場合に比べて何倍速いか，「付加データ」は高速化のために追加される付加データの大きさは入力データの何%程度か，「前処理」は前処理にかかる時間が2009年現在の高性能なパソコンでどのくらいか，を表す．

3. A* 法

A*（A-star）法は各点 $v\in V$ と終点 $t\in V$ との最短路長の下界 $\overline{d(v,t)}$ を用いて，探索点の数を節約する

表1　各最短路検索手法の比較

手法	検索	付加データ	前処理
A*+ランドマーク	数十倍	数百%	数十秒
ビットベクトル	数百倍	数百%	数日
ハイウェイ	数万倍	百数十%	数十分
トランジット	数十万倍	数百%	数時間
階層メッシュ	数千倍	数%	数時間

みやもと　ゆういちろう
上智大学 理工学部
〒102-8554 千代田区紀尾井町 7-1
ふじさわ　かつき
中央大学 理工学部
〒112-8551 文京区春日 1-13-27
くぼ　みきお
東京海洋大学 海洋工学部
〒135-8533 江東区越中島 2-1-6

方法である[2][4].

3.1 A* 法のアルゴリズム

枝 $(v,w)\in E$ の重みを $l(v,w)$ とする．始点 s から点 $v\in V$ への最短距離の暫定値を $d(v)$ とする．アルゴリズム終了時の $d(t)$ が st-最短路の長さである．

ダイクストラ法との違いは4行目と12行目である．4行目ですべての $v\in V$ に関して $\overline{d(v,t)}:=0$ とすれば，ダイクストラ法と挙動が同じになる．A* 法とダイクストラ法を比べた場合，A* 法の長所は探索点を少なくできること，短所は一度探索した点を再び探索する可能性があることと一探索あたりの計算が（下界を考慮する分）若干複雑であることである．

Algorithm 1 A*法

```
1: d(s) := 0, d(v) := ∞, ∀v ∈ V \ {s}
2: S := V
3: while S ≠ ∅ do
4:    d(v) + d(v,t) が最小の点 v ∈ S を選択
5:    if v = t then
6:       finish
7:    end if
8:    S := S \ {v}
9:    for all (v,w) do
10:      if d(v) + l(v,w) < d(w) then
11:         d(w) := d(v) + l(v,w)
12:         S := S ∪ {w}
13:      end if
14:   end for
15: end while
```

3.2 ランドマークを用いた下界の強化

例えば，入力データのネットワークがユークリッド平面上に埋め込まれたもので，枝重みが枝の両端点のユークリッド距離であるとき，点対間のユークリッド距離は点対間の最短路長の下界となる．この下界をA* 法で利用するのはもっとも単純な工夫の一つである．より強いA* 法の下界として，**ランドマーク**（landmark）とよばれる点の集合 $K\subseteq V$ を用いる方法がある[3]．一般に枝重み付きネットワーク $G=(V,E)$ の3点 $k,u,v\in V$ に関して，三角不等式 $d(u,v)\geq d(k,v)-d(k,u)$ が成り立つ[1]．ランドマークを用いた工夫では，すべての $k\in K$，すべての $v\in V$ に関して $d(k,v)$ を前処理で計算し $\overline{d(v,t)}:=\max\{d(k,t)-d(k,v)\mid k\in K\}$ とする．図1にダイクストラ法およびA* 法で探索される範囲の例を示す．最短路は右下（始点）から左上（終点）への太線である．薄いものから順に，ダイクストラ法で探索される範囲，下界としてユークリッド距離を用いたA* 法で探索される範囲，ランドマーク16個を用いたA* 法で探索される範囲である．ランドマークをまばらに配置すれば，その数は十数個で十分な性能を発揮することが知られている．このとき付加データの大きさは入力データの数倍程度である．ランドマークを用いる長所は付加データの生成が，最短路木をランドマークの数だけ作ればよいので，かなり高速であることである．短所は，ランドマークをいくら増やしても検索時の計算時間はあまり減らないことである．

[1] これは最短路長に関する三角不等式なので，枝重みに関して三角不等式が成り立たなくても，必ず成り立つ．

4. ビットベクトル法

ビットベクトル（bit vector）**法**は，最短路を探索する際の探索方向を終点方向に強く限定する手法である．そのために付加データとして各枝にバイナリベクトルを対応づける[5]．前処理では，まず入力ネットワーク $G=(V,E)$ の点集合 V を N 個の部分集合 $V_1,\cdots,V_N$ に分割する（ここで N はビットベクトル法を調整するためのパラメータである）．そしてネットワークの各枝 $e\in E$ に付随するベクトルの第 i 成分を「終点が V_i に含まれる最短路に e が含まれるならば1，そうでないならば0」に設定する．検索時には，始点からダイクストラ法で最短路を探索するが，終点が V_j に含まれるならば，付随するベクトルの第 j 成分が1である枝のみを探索に用いればよい．

分割 $V_1,\cdots,V_N$ の各部分集合 V_i が近い場所にまとまっていると枝に付随するベクトルの1成分が少なく

図1 ダイクストラ法とA* 法の探索範囲

なるため効率がよい．よって一般には（超）平面に埋め込まれた点集合を平面上の矩形分割に対応して分割することが多い．点集合 V の分割数が N のとき，枝に付随するベクトルはそれぞれ N 次元ベクトルになる．よって，例えば32ビットコンピューターであれば N は32の倍数が適切である．N が数十でも十分な性能を発揮する．

長所は，パラメータ N の設定範囲が広いことである．N が点数に近くなると，各点に関して（終点に向かう）最短路木を保持することに近くなる．短所は，前処理時間の短縮が難しいことである．

5. ハイウェイヒエラルキー法

ハイウェイヒエラルキー（highway hierarchy）**法**は，最短路でよく使われる枝（ハイウェイとよばれる）を階層的に抽出しておいて，検索時にそれを用いる方法である[7]．

点 $v\in V$ の近傍 $N(v)$ を「v に最も近い x 点」と定義する（ここで x はハイウェイヒエラルキー法を調整するためのパラメータである）．枝 $e\in E$ は，それを含む st-最短路が存在して $N(s)$ にも $N(t)$ にも接続しないとき，ハイウェイであると見なされる．

検索時には始点から x 点までは入力データを，それ以降は付加データであるハイウェイのみを探索に使う．終点付近でも入力データを使う必要があるため，双方向探索をする必要がある．

また，ハイウェイのみからなるネットワークからさらに上位のハイウェイを同様に抽出することによって階層的なハイウェイを構築し，大幅な効率化を図れる．近傍の点数は数十程度，ハイウェイの階層は数層で十分な性能を発揮する．長所は，定義通りに計算すると時間がかかりそうなハイウェイの抽出（前処理）が，アルゴリズムの工夫により高速にできることである．短所は，双方向探索しかできないことと，それに伴い探索が若干複雑であることである．

6. トランジットノード法

トランジットノード（transit node）法は「遠い地点間を結ぶ最短路の多くは重要な交差点を通る」という経験則に基づいた手法である．

トランジットノードの集合 T を「ある程度長い最短路は必ず T のいずれかを通るような $T\subset V$」と定義する．トランジットノードの集合 T をひとたび定めると，ある程度長い st-最短路上にはトランジットノードが複数含まれることがある．これらのトランジットノードのうち，始点 s から最初に訪れる可能性があるものの集合を $T_{\text{out}}(s)$ で表す．同様に終点 t の直前に訪れる可能性があるものの集合を $T_{\text{in}}(t)$ で表す．前処理では，各点 v に対応する $T_{\text{out}}(v)$，$T_{\text{in}}(v)$ の大きさが平均的に定数程度で抑えられるような，そして T の大きさもなるべく小さくなるようなトランジットノードの集合を見つける．そしてすべての $v\in V$ に関して $d(v,w)$，$\forall w\in T_{\text{out}}(v)$ および $d(w,v)$，$\forall w\in T_{\text{in}}(v)$ を計算・記憶し，すべての $v,w\in T$ に関して $d(v,w)$ を計算・記憶する．

検索時には，与えられた s，t に対して，$\min\{d(s,v)+d(v,w)+d(w,t)\mid v\in T_{\text{out}}(s), w\in T_{\text{in}}(t)\}$ を計算すれば st-最短路長がわかる（ただし s と t が近い場合にはトランジットノードを利用できない）．$T_{\text{out}}(v)$，$T_{\text{in}}(v)$ の大きさが平均的に定数程度に抑えられていれば，検索における計算は非常に短い時間で終了する．

付加データの大きさはトランジットノードの数に依存し，おおむね $|T|^2+|V|\times(T_{\text{out}}(v),T_{\text{in}}(v)$ の平均）程度である．トランジットノードの集合 T の大きさは数百から数千で十分な性能を発揮する．このとき付加データの大きさは入力データの数倍程度になる．

欠点は，あまり長くない st-最短路ではトランジットノードを利用できないことである．トランジットノードを利用できない場合には，別の手法で st-最短路を見つけなければならない．

ハイウェイヒエラルキー法の最上位ハイウェイの点をトランジットノードとする手法が提案されており，良好な結果が報告されている[1]．

7. 階層メッシュ疎化法

階層メッシュ疎化法は，遠い地点間の最短路では絶対に使われない枝を省いた疎なネットワークを前処理で生成し，それを検索時に利用して高速化する手法である[6]．前処理および検索においてグラフの点の座標を用いることが特徴であり，結果として付加データの軽減に成功している．付加データの大きさは入力データの数%で十分な性能を発揮する．以下，グラフは（超）平面に埋め込まれているとする．

前処理では，平面を矩形分割し，各矩形に含まれる枝のうち，その矩形から遠いところ同士を結ぶ最短路で使われない枝を省く．例えば，2次元平面 $\mathbb{R}^2$ を一辺の長さ c の正方形領域 $R_{i,j}:=\{(x,y)\in\mathbb{R}^2\mid ic\le x<(i+1)c, jc\le y<(j+1)c\}$ に分割する．そして，領域

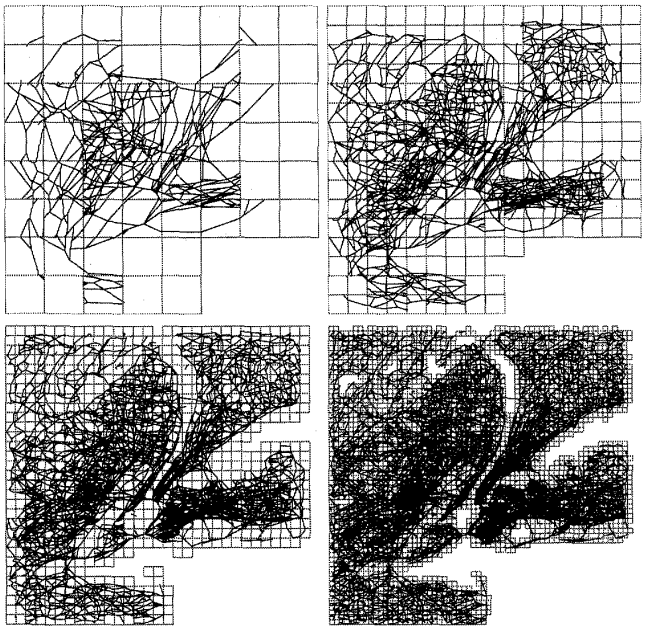

図2　抽出された枝の例

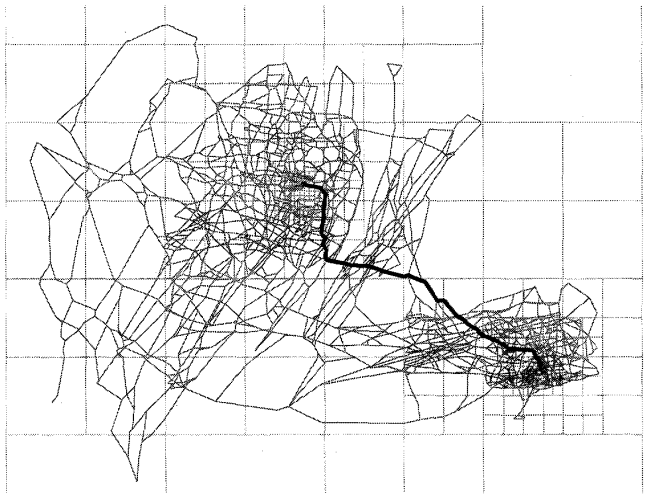

図3　疎化ネットワーク上の最短路の例

$R_{i,j}$ に含まれる枝のうち，領域 $\{(x,y)\in\mathbb{R}^2 \mid (i-1)c\leq x<(i+2)c, (j-1)c\leq y<(j+2)c\}$（これは $R_{i,j}$ を中心として9倍の面積を持つ正方形である）の外側に始点・終点をもつ最短路では一度も使われないものを省く．正方形が大きければ，残された枝は入力データに比べて疎なものとなる．図2は残った枝を正方形の大きさごとに示した例である．この例では，正方形の一辺の長さを2の冪乗に限定している．これにより，大きな正方形で残される枝の計算に，小さな正方形で残された枝のみを利用できるので前処理が効率的に行える．

検索時には，始点・終点に近いところでは小さな正方形に含まれている枝を，遠いところでは大きな正方形に含まれている枝のみを使って最短路探索すればよい．図3に階層メッシュ疎化法の検索で用いられる疎なネットワークの例を示す．

長所は，疎なネットワークを抽出しているだけなので，検索時の探索法の選択に自由度があること，それゆえに実装が簡単であることである．短所は，グラフを（超）平面に埋め込む必要があること，前処理時間の短縮が難しいことである．

参考文献

[1] Bast, H., Funke, S., Sanders, P. and Schultes, D.: Fast Routing in Road Networks with Transit Nodes, *Science*, Vol. 316 (2007), 566.

[2] Doran, J.: An approach to automatic problem-solving, *Machine Intelligence*, Vol. 1 (1967), 105-127.

[3] Goldberg, A. V. and Harrelson, C.: Computing the shortest path: A search meets graph theory, in *Proceedings of the 16th annual ACM-SIAM symposium on Discrete algorithms*, 2005, SIAM.

[4] Hart, P. E., Nilsson, N. J. and Raphael, B.: A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions on System Science and Cybernetics*, Vol. 4 (1968), 100-107.

[5] Köhler, E., Möhring, R. H. and Schilling, H.: Acceleration of shortest path and constrained shortest path computation, in *Experimental and Efficient Algorithms*, Vol. 3503 of *LNCS*, Springer, 2005.

[6] 宮本裕一郎, 宇野毅明, 久保幹雄：最短路高速検索のための階層メッシュ疎化法，情報処理学会研究報告，第AL-119巻, 2008.

[7] Sanders, P. and Schultes, D.: Highway hierarchies hasten exact shortest path queries, in *Proceedings of the 13th European Symposium on Algorithms*, Vol. 3669 of *LNCS*, Springer, 2005.