OR 事典 Wiki

最短路問題

藤澤　克樹，宮本裕一郎，久保　幹雄

最短路問題は経路探索などの多くの応用を持ち，また他の最適化問題の子問題として用いられることも多く，適用範囲の広い組合せ最適化問題である．そのため，最短路問題を高速に解くことは重要な意味を持つ．最短路問題に対する解法には，安定的かつ効率的な高速アルゴリズムが存在するが，実問題は大規模（全米の道路ネットワークでは，2400万点，5800万枝にも及ぶ）になるため高速化が不可欠である．本解説では最短路問題の定義・種類を述べた後，著名なアルゴリズムであるダイクストラ法に対する高速実装と実験結果等について説明を行う．

1．最短路問題の定義と種類

最短路問題は次のような問題である．

> ある有向グラフ $G=(V,E)$ と各枝 $(v,w)\in E$ に重み $l(v,w)$ が付与されているネットワーク $N=(G,l)$ が与えられているとする．このとき有向グラフ G の任意の2点 $s,t\in V$ に対し，点 s を始点とし，点 t を終点とする有向パス p の中で，その長さ（パスに含まれる枝の重みの合計）が最小のパスを点 s から点 t への最短路（shortest path）といい，最短路を求める問題を最短路問題（shortest path problem）という．

最短路問題は次のように3種類に分けることができる．

● 1対全（single-source）最短路問題

1点から全点に対する最短路と最短距離（最短路の長さ）を求める問題．非負の枝の重みを持つグラフ上ではダイクストラ（Dijkstra）法が，負の重みを持つ枝がある場合にはベルマン・フォード（Bellman-Ford）法が有名である．

●全対全（all-pairs）最短路問題

全点間の最短路と最短距離を求める問題．ワーシャル・フロイド（Warshall-Folyd）法が有名であるが，1対全最短路問題アルゴリズムを各点に対して適用することでも求めることができる．

● 1対1（point-to-point）最短路問題

2点間の最短路と最短距離を求める問題．

2．探索アルゴリズムと実装方法

最短路問題に関する探索アルゴリズムの中でも，最も頻繁に利用されるダイクストラ法を取り上げ，アルゴリズムの説明から，高速化のための優先キューの適用方法・実装方法について示していく．

2.1 ダイクストラ法

1959年，E. W. Dijkstra[1]によって提案された1対全最短路問題に対するアルゴリズムである．探索するグラフ上のすべての枝の重みが非負でなければならないが，ベルマン・フォード法[2]に比べ効率的かつ高速である．ダイクストラ法は，ベルマン・フォード法と同じように動的計画法の一種とみなすことができるが，点の探索の順序に特徴をもつ．探索の途中で得られた始点 s からの距離の最小値をポテンシャルもしくは距離ラベルとよび，その値の小さい順に点の探索を行い，最短路および最短距離を確定していく．つまり，各点に対し，$d(s)=0, d(v)=+\infty (v\in V)$ とするポテンシャル d を用意し，探索点 v に接続する枝 (v,w) に対して，$d(v)+l(v,w)<d(w)$ ならば $d(w):=l(v,w)+d(v)$ とする操作を繰り返し行う．終点 t が探索点になった時点で終了させることで，1対1最短路問題も効率的に求解可能である．

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
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
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

Algorithm 1 ダイクストラ法

```
1: ポテンシャル d の初期設定
2: S := V
3: while S ≠ ∅ do
4:    d(v) が最小の点 v ∈ S を選択
5:    S := S\{v}
6:    for all (v, w) do
7:       if d(v) + l(v, w) < d(w) then
8:          d(w) := d(v) + l(v, w)
9:       end if
10:   end for
11: end while
```

2.2 高速化のための優先キューの利用

ダイクストラ法は，効率的かつ高速なアルゴリズムだが，探索点を決定する操作（Algorithm 1の4行目，5行目）が，ボトルネックであることが知られている[1]．探索候補点集合（Algorithm 1のS）に対して優先キューを適用することで，改善できる．

2.3 優先キューの実装方法

優先キューとは，データを優先度によって操作するデータ構造のことである．ダイクストラ法に適用する優先キューは，insert, decrease-key, extract-minと呼ばれる操作に対応している必要がある．

- insert：ポテンシャル$d(v)$を優先度として，点$v \in V$を優先キューQへ挿入する．
- decrease-key：優先キュー内に格納されている点$v \in Q$の優先度$d(v)$を，$d'(v)$に更新する．
- extract-min：優先度$d(v)$が最小である点$v \in Q$を取り出し，点vを次探索点とする．

ダイクストラ法では，繰り返しこれらの操作を行うことになるため，より効率的なものを適用することで，高速化が可能になる．

2.4 データ構造の選択

優先キューは，木構造による安定した実行が可能なヒープ系データ構造と，データをルールに従いバケットに格納するバケット系データ構造とに分類が可能である．ここでは，それぞれの最も基本的なデータ構造であるバイナリ・ヒープと1レベル・バケットに加え，実験的に最も高速であるといわれているマルチレベル・バケットを取り上げ，比較を行う．

2.4.1 2-heap（バイナリ・ヒープ）

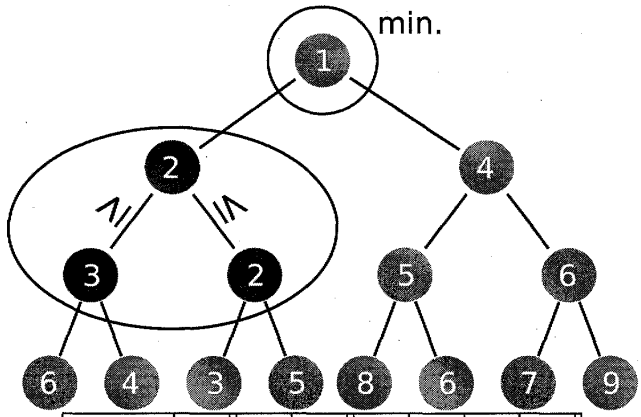
2-heap[3]は，図1のように，親は高々2つの子を持つ木構造で構成される．常に親の優先度は子より高い．点数n，枝数mに対して，1対全最短路問題の計算量は$O((m+n)\log n)$である．

2-heapは以下の特徴をもつ．

1. 木構造により実行性能が安定している．
2. メモリ要求量が大変少なく，データの局所性を高めやすい．
3. 木構造のためデータの読み書きが多発するので，データ交換操作がボトルネックとなりやすい．

2.4.2 Buckets（1レベル・バケット）

Buckets[4]は，最大枝長Uに対し，$U+1$個のバケットの循環リストとして構成される（図2）．点vのポテンシャル$d(v)$に対し，$d(v)\bmod(U+1)$により操作するバケットを決定する．各バケット内のデータは双方向リストで結ばれており，同一バケット内に格納されている点の優先度は常に等しい．1対全最短路問題の計算量は，点数n，枝数m，最大枝長Uに対し，$O(m+nU)$である．



Index	0	1	2	3	4	5	6
Data	1	2	4	3	2	5	6

Index	7	8	9	10	11	12	13	14
Data	6	4	3	5	8	6	7	9

図1 バイナリ・ヒープの例

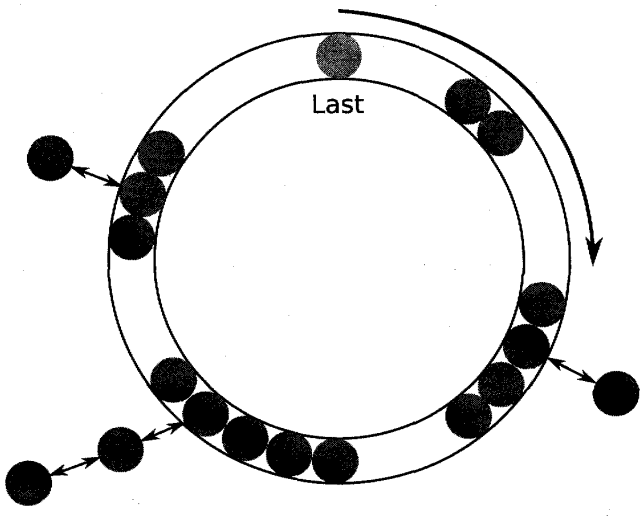


図2 1レベル・バケットの例

1．実行時間・メモリ要求量が，最大枝長に依存する（重み小：高速，重み大：低速）．

2．道路ネットワークと相性が良い．

2.4.3 MLB（マルチレベル・バケット）

MLB (Multi-Level Buckets)[5]は，ヒープ系優先キュー，バケット系優先キューの双方の有効な特性を融合させた優先キューである．グラフ特性の影響は小さく安定かつ高速であるが，非常にメモリ要求量が大きい．1レベル・バケットとは異なり，MLBではバケットは$\lceil \log U \rceil+1$だけ用意すればよい．点数n，枝数m，最大枝長Uに対し，1対全最短路問題の計算量は$O(m+n\log U)$となる．

1．グラフ特性の影響は小さく，安定かつ高速

2．バケットの決定には高速なビット演算で行う

3．メモリ要求量が非常に大きい

2.4.4 優先キューの比較

点数n，枝数m，最大枝数Uであるグラフに対して，計算量をまとめると表1のようになる．1対全最短路問題では，枝数分のinsertとdecrease-key，点数分のextract-minを行うことになる．また，その特性は図3となる．これは，比較的小さいグラフに対し，各枝長を$2^t(t=[-10,-9,\cdots,9,10])$倍し，計算機実験を行った結果である．Bucketsの特性がよく表れている．

表1　優先キューの計算量

	insert	decrease-key	Extract-Min
2-heap	$O(\log n)$	$O(\log n)$	$O(\log n)$
Buckets	$O(1)$	$O(1)$	$O(U)$
MLB	$O(1)$	$O(1)$	$O(\log U)$

	1対全最短路問題の計算量
2-heap	$O((n+m)\log n)$
Buckets	$O(m+nU)$
MLB	$O(m+n\log U)$

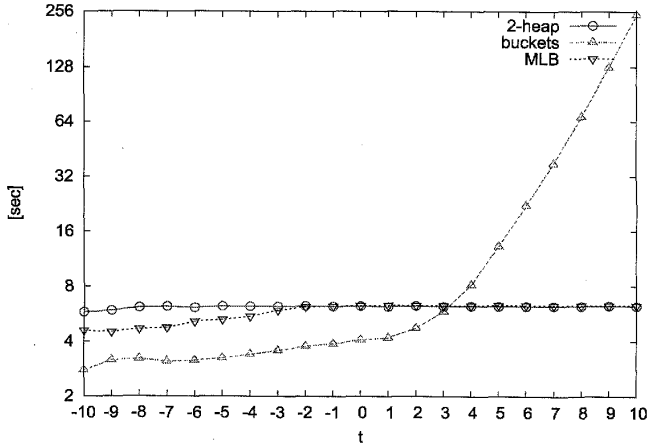


図3　優先キューの特性

2.5 優先キューの性能

2-heap, Buckets, MLBそれぞれの全米道路ネットワークに対しての計算機実験を行い，実行時間とメモリ要求量をまとめると，表2，表3のようになる．用いたグラフデータは，約2400万点，約5800万枝にも及ぶ道路ネットワークを変換させた大規模な実データである（TIGER/Line[1]で公開されている）．実行時間は，ランダムに選択した1対1最短路問題を256回解いた際の平均である．

最短路問題では計算量に対してデータ量が大規模であるため，データ型によるメモリ使用量や実行時間に対する影響が大きい．各変数が32ビット整数型と64ビット変数型の2種類の計算機実験の結果を記載する．グラフデータは大規模であるが，32ビット整数型の範囲で収まるため，どちらの場合でもアルゴリズムは正しく終了するが，距離ラベルはいずれも64ビット整数である．

3. 大規模最短路問題に対する高速処理システム

現在ではWeb上で最短経路を求めるサービスが複数行われている（Google Maps[2]や最短路問題オンラインソルバー[3][6]など）．

表2　全米グラフの実行時間（単位は秒）

	32ビット整数型	64ビット整数型
2-heap	2.47	3.14
Buckets	1.68	2.28
MLB	2.63	2.65

表3　全米グラフのメモリ要求量（単位はGB）

	32ビット整数型	64ビット整数型
2-heap	1.27	1.99
Buckets	1.09	1.62
MLB	2.17	2.17

[1] http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html

[2] http://maps.google.com/

[3] http://opt.indsys.chuo-u.ac.jp/portal/

これらのシステムでは，多くのユーザからの要求に対応する必要があるので，単にダイクストラ法実行1回の処理を高速化するだけでなく，それらを並列に高速処理する技術が必要である．

ユーザからの要求量は時間帯や突発的な事象によって大きく変動するため，最大需要を見越した計算資源を保有しておくのは無駄が多くコストも増大してしまう．そこでクラウド・コンピューティング技術を用いて，高負荷時には自動的に計算機資源の増大を行うシステムなどが必要になってくる．

最近ではカーナビゲーション・システムの拡張として渋滞・事故情報をリアルタイムに把握しながら集約された大規模サーバで最短路を探索してユーザに結果を送信するシステムも構築されている．動的な情報を考慮した最短路探索を行うことにより，これまでの経路探索に比べてより精度を向上させることができる．さらに特定の道路への集中を防ぐように交通量の分散を行うことで，渋滞の緩和，事故発生率の低下，排出ガスの削減などを目指した交通管制を行うことも可能になると予想されている．

参考文献

[1] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, 1: 269-271, 1959.

[2] R. E. Bellman, "On a Routing Problem," Quarterly of Applied Mathematics, 16: 87-90, 1958.

[3] J. Williams, "Heapsort," Communications of the ACM, 7: 347-348, 1964.

[4] R. B. Dial, "Algorithm 360: Shortest Path Forest with Topological Ordering," Comm. ACM, 12: 632-633, 1969.

[5] A. V. Goldberg, "A Simple Shortest Path Algorithm with Linear Average Time," Technical Report STAR-TR-01-03, STAR Lab., InterTrust Tech., Inc., Santa Clara, CA, USA, 2001.

[6] 安井雄一郎，藤澤克樹，笹島啓史，後藤和茂，宮本裕一郎，"大規模最短路問題に対するダイクストラ法の高速化，" 2008年度日本OR学会秋季研究発表会，314-315, 2008.