数理解析研究所講究録
第1799巻 2012年 92-99

格子グラフ上の最短経路問題のための劣線形領域アルゴリズム

今井 達也* 野口 俊輔† 藤 哲郎†

概要

本論文では，格子グラフ上の二頂点間の最短経路を頂点数 n の劣線形サイズの作業領域で求めるアルゴリズムについて述べる．提案アルゴリズムは，自然数パラメータ l をとって，$O(n^{\frac{2^l}{2^{l+1}-1}})$ の空間計算量，および $O(n^{\frac{2^l(l+2)}{2^{l+1}-1}})$ の時間計算量で動作する．

1 序論

グラフの最短経路問題は数多くの応用を持つ非常に重要な問題の一つである．これまでに，理論的なものや汎用的なものから，問題固有の性質を利用したヒューリスティックなものまで，数多くの最短経路アルゴリズムが提案されてきた．

近年では社会の複雑化に伴って，より巨大なグラフに対する最短経路問題の解決が求められるようになってきている．しかし，巨大なグラフに対する最短経路問題を実際に計算機の上で解くためには，グラフの大きさに伴った多くのメモリが必要になる．これまでに提案されてきた最短経路アルゴリズムはグラフの頂点数に比例する空間計算量を必要とするものが多く，劣線形領域のアルゴリズムはほとんど知られていない．そのような中にあって，Asano と Doerr は格子グラフに対する劣線形領域の最短経路アルゴリズムを提案した [1]．このアルゴリズムは，自然数パラメータ l をとり，グラフの頂点数 n に対して入力グラフや得られた最短経路を表すために使用される領域を除いて[1]，$O(n^{\frac{l+1}{2^l+1}})$ の空間計算量，および $O(n^{2+\frac{1}{2}-\frac{l(l+1)}{4(2^l+1)}})$ の時間計算量で動作するアルゴリズムである．

本研究では，Asano らのアルゴリズムを元に，同じく自然数パラメータ l をとって，$O(n^{\frac{2^l}{2^{l+1}-1}})$ の空間計算量，および $O(n^{\frac{2^l(l+2)}{2^{l+1}-1}})$ の時間計算量を持つ最短経路アルゴリズムを提案した．以下，Asano らのアルゴリズム，我々のアルゴリズムの順に，その詳細を述べていく．なお以降の全ての議論では，入力される格子グラフが正方形状であることと，辺のコストが全て有限の正の値であること，始点が格子グラフの左下端，終点が右上端の頂点であることを仮定して話を進めていく．これらの仮定は [1] で述べられている方法によって容易に緩和できるが，本論文ではその方法については言及しない．

2 Asano, Doerr の非再帰版省メモリ最短経路アルゴリズム

Asano らのアルゴリズムは，Dijkstra 法 [2] などと同様に，始点から各頂点までの最短距離を計算していき，それを利用して始点から終点への最短経路を求めるアルゴリズムである．このアルゴリズムは，特定の

*東京工業大学 大学院情報理工学研究科 数理・計算科学専攻
†東京工業大学 理学部 情報科学科

[1]入力が書き込み不可能なメモリに保存されている場合や，入力が，格子グラフの縦横の頂点数と，辺を与えるとそのコストを返す関数の対によって与えられている場合などを想定している．また，出力に関しては，最短経路が通過する頂点を順次画面に書き出して捨ててしまうような場合を想定している．

頂点への最短距離だけを覚えておくことによって使用メモリの削減を達成した．より具体的には，最短距離が保存される頂点は，自然数パラメータ $k \in \mathcal{N}$ に基づいて図 1 のように k^2 個に分割された小格子グラフ $(S[1], \cdots, S[k^2])$ のふち部分の頂点（黒丸で表された頂点）である．以降は黒丸の頂点を**境界頂点**と呼ぶ．境界頂点は複数の小格子グラフに含まれる場合がある．

以下，始点からの最短距離を求める部分と最短経路を求める部分とに分けてアルゴリズムを解説する．

2.1 最短距離を求めるアルゴリズム

入力された格子グラフを $G = (V, E)$，始点を $s \in V$，グラフの一辺の分割数を $k \in \mathcal{N}$ とおく．また，頂点 $v \in V$ から頂点 $v' \in V$ への最短距離を $d(v, v')$ とおく．

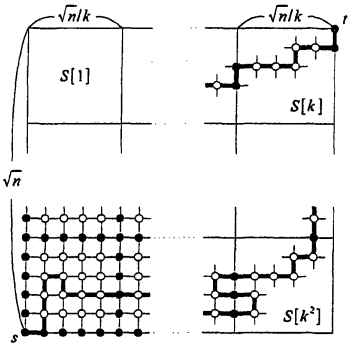

図 1: k^2 個に分割された格子グラフ（太線は最短経路の一例）

このアルゴリズムはまず，s から各境界頂点への最短距離（の上界）を保存する配列 C を用意し，各要素を十分大きな数で初期化する（$C[s]$ は 0）．次に k^2 個の小格子グラフそれぞれに対して，小格子グラフの各境界頂点を始点とする多始点最短経路アルゴリズムを走らせる．使用する多始点最短経路アルゴリズムは Dijkstra 法の応用であり，厳密には次の手順で行われる．まず始めに Dijkstra 法と同様に，小格子グラフに含まれる各頂点への最短距離を保存する配列 T を用意し，各要素を十分大きな数で初期化する．次に，始点，すなわち対象としている小格子グラフの各境界頂点の T の値を C の値で置き換える．後は Dijkstra 法の前半部分と同様に，最も小さな未確定の最短距離を持つ頂点を一つ選び，その頂点の最短距離を確定，隣接頂点に距離を伝搬する，という処理を頂点の個数だけ繰り返す．全ての頂点への最短距離が求まったら，最後に各境界頂点の T の値を C に書き入れる．k^2 個のグラフに対して上記のアルゴリズムを走らせるこの処理を**走査**と呼ぶ．また本論文では走査中の各小格子グラフに対する上記の処理を**多始点版 Dijkstra 法**と呼ぶこととする．各走査の際の小格子グラフの選択順序に依らず，回数が高々境界頂点の個数に達するまで走査を繰り返すと，C に各境界頂点への最短距離が正しく定まる．

このアルゴリズムの擬似コードを Algorithm 1 に示す．なお，boundary(G) は格子グラフ G のふち部分の頂点の集合を表す．また，Dijkstra_distance(G, T) は，格子グラフ G に対して Dijkstra 法の前半部分を走らせる処理である．T は最短距離の上界を保存するための既に初期化された配列である．

Algorithm 1 最短距離を求めるアルゴリズム AD_distance(G, s, k)

```
1: ∀v ∈ ⋃_i boundary(S[i]), C[v] ← ∞;
2: C[s] ← 0;
3: for 1, ..., |⋃_i boundary(S[i])| do
4:   for j ← 1, ..., k^2 do
5:     ∀v ∈ V(S[j]), T[v] ← ∞;
6:     ∀v ∈ boundary(S[j]), T[v] ← C[v];
7:     Dijkstra_distance(S[j], T);
8:     ∀v ∈ boundary(S[j]), C[v] ← T[v];
9:   end for
10: end for
11: return C;
```

このアルゴリズムの正しさは次の理由からなる．まず，任意の境界頂点 v について，入力グラフに対する仮定より明らかに，s から v までの最短経路が少なくとも一つ存在する．議論を簡単にするため，ここではそれぞれの境界頂点への最短経路は一つずつしかないと仮定しよう．s から v への最短経路と各格子グラフの境界頂点の交点を s から近い順に $v_0, v_1, v_2, \ldots, v_t (v_0 = s, v_t = v)$ とすると，$C[v_i] = d(s, v_i)$ かつ $C[v_{i+1}] > d(s, v_{i+1})$ のとき，v_i と v_{i+1} の両方を含む小格子グラフに対して多始点版 Dijkstra 法を走らせれば，$C[v_{i+1}] = d(s, v_{i+1})$ となる．全ての境界頂点への最短距離がまだ求まっていない場合，上記の v_i と v_{i+1} の関係にある頂点が必ず一つ以上存在する．よって走査の中で多始点版 Dijkstra 法を走らせる小格子グラフの順序に依らず，走査を行うたびに少なくとも一つの頂点の最短距離が新たに求まる．従って高々境界頂点の個数だけ走査を行えば全ての境界頂点への最短距離が C に定められる．

本節の最後に，各多始点 Dijkstra 法では力任せの方法で次に値を決定する頂点を探すものとして，このアルゴリズムの計算量を求める．このアルゴリズムの時間計算量は，アルゴリズムの定義より，走査の時間計算量の境界頂点数倍である．多始点版 Dijkstra 法の時間計算量は Dijkstra 法と同等であるため，各小格子グラフに $O(n/k^2)$ 個の頂点が存在することから，多始点版 Dijkstra 法の時間計算量は $O(n^2/k^4)$ である．よって，一回の走査の時間計算量は $O(n^2/k^2)$ である．各小格子グラフには $O(\sqrt{n}/k)$ 個の境界頂点が含まれ，グラフ全体の境界頂点の数は $O(k\sqrt{n})$ であるので，このアルゴリズム全体の時間計算量は $O(n^2\sqrt{n}/k)$ となる．また，配列 C の要素数は $O(k\sqrt{n})$，T の要素数は $O(n/k^2)$ であることから，このアルゴリズムの空間計算量は $O(k\sqrt{n} + n/k^2)$ となる．

使用メモリの削減に主眼をおく場合，空間計算量を最小化する k を k^* とおくと，$k^* = \Theta(n^{1/6})$ となる．$k = n^{1/6}$ を代入すると空間計算量は $O(n^{2/3})$ となり，時間計算量は $O(n^{2+1/3})$ となる．

2.2 最短経路を出力するアルゴリズム

$G = (V, E), s \in V, k \in \mathcal{N}, d$ は 2.1 節と同様に定義し，G, s, k の他に終点 $t \in V$ を入力とする．

このアルゴリズムは，s から t へのある最短経路上のいくつかの境界頂点 $v_0, v_1, \ldots, v_t (v_0 = s, v_t = t)$ に対して，v_t から v_{t-1} への最短経路，v_{t-1} から v_{t-2} への最短経路，..., v_1 から v_0 への最短経路，と境界頂点の間の最短経路を順に出力していくアルゴリズムである．t から v_i までの経路が出力されているとき，v_{i-1} の決定，および，v_i から v_{i-1} への最短経路の出力は次のようにして行われる．まず，v_{i-1} は，$d(s, v_{i-1}) + d(v_i, v_{i-1}) = d(s, v_i)$ を満たし，v_i を含むいずれかの小格子グラフに含まれていなければならない．よって，このアルゴリズムはまず，v_i を含む各小格子グラフの境界頂点 v に関して $d(s, v)$ を計算するために，AD_distance(G, s, k) を走らせる．次に，$d(v_i, v)$ を計算するために v_i を含む各小格子グラフに対して v_i を始点として Dijkstra 法の前半部分を走らせる．そして上記の式を満たす中で $d(v_i, v)$ が最も大きい v を v_{i-1} に定め，Dijkstra 法と同様のやり方で v_i から v_{i-1} への最短経路を出力する．このとき，出力される経路の向きを考えて，$d(v_i, v)$ を求める際に計算した値を捨てて最短距離の計算もやり直さなければならない．

以上をまとめ，このアルゴリズムの疑似コードを Algorithm 2 に示す．ただし，Dijkstra(G, s, t) はグラフ G の頂点 $s \in V(G)$ から $t \in V(G)$ への最短経路を Dijkstra 法に基づいて出力するサブルーチンである．

用いられる配列およびその要素数は AD_distance と同じであることから，このアルゴリズムの空間計算量は $O(k\sqrt{n} + n/k^2)$ であり，k^* を代入するとこの式は $O(n^{2/3})$ となる．また，時間計算量 $O(n^2\sqrt{n}/k)$ の AD_distance および $O(n^2/k^4)$ の Dijkstra_distance, Dijkstra を高々 $O(k\sqrt{n})$ 回繰り返すため，このアルゴリズムの時間計算量は k によらず $O(n^3)$ である．

Algorithm 2 最短経路を出力するアルゴリズム AD_path(G, s, t, k)

```
1: while t ≠ s do
2:   C ← AD_distance(G, s, k);
3:   for t ∈ boundary(S[i]) を満たす i do
4:     ∀v ∈ V(S[i]), T[v] ← ∞; T[v] = 0;
5:     Dijkstra_distance(S[i], T);
6:     if C[v'] + T[v'] = C[t] を満たす v' ∈ boundary(S[i]) が存在する then
7:       v ← 上記 v' ∈ boundary(S[i]) の中で T[v'] が最も大きい v';
8:       j ← i;
9:     end if
10:  end for
11:  Dijkstra(S[j], v, t);
12:  t ← v;
13: end while
```

3 提案アルゴリズム

2節のアルゴリズムでは，境界頂点への最短経路を記憶する配列と，小格子グラフ一つ分の頂点数と等しいサイズの作業領域だけを使用することによって，空間計算量が削減されている．Asano らは [1] において，2節のアルゴリズムに加え，AD_distance で使われている Dijkstra_distance の代わりに再帰的に Asano らのアルゴリズムを適用することで，作業領域を更に削減し，より少ないメモリで最短距離を求めるアルゴリズムを提案した．

この先行研究を基に，我々は次の改善を行うことによって Asano らのアルゴリズムよりも時間・空間計算量の効率が良い最短経路アルゴリズムを提案した．

- 非再帰のアルゴリズムの時間計算量を改善．
- 再帰時の格子グラフの分割方法を改善．

以下，それぞれのアルゴリズムを順に解説する．

3.1 高速に最短距離を求めるアルゴリズム

Asano らの始点から各境界頂点への最短距離を求めるアルゴリズムは，Dijkstra 法と同様に始点からの最短距離が短い順に最短距離を定めていくことによって高速化することができる．より正確には，最初に始点から始点への最短距離を 0 で初期化し，

1. 始点からの最短距離が未確定の境界頂点のうち最小のもの $v_{\min}$ を探す．始点から $v_{\min}$ への最短距離は現在の値で確定する．
2. $v_{\min}$ を含む全ての小格子グラフに対して多始点版 Dijkstra 法を適用し，それらの小格子グラフに含まれる境界頂点への最短距離を更新する．

という処理を境界頂点の個数だけ繰り返すことによって，Asano らのアルゴリズムよりも高速に始点から各境界頂点までの最短距離を正しく求めることができる．このアルゴリズムの擬似コードを Algorithm 3 に示す．

Algorithm 3 より高速に最短距離を求めるアルゴリズム (G, s, k)

1: $\forall v \in \bigcup_i \mathrm{boundary}(S[i])$, $C[v] \leftarrow \infty$; $C[s] \leftarrow 0$;
2: $\forall v \in \bigcup_i \mathrm{boundary}(S[i])$, $D[v] \leftarrow$ false;
3: **for** $1, \ldots, |\bigcup_i \mathrm{boundary}(S[i])|$ **do**
4: $v_{\min} \leftarrow D[v] =$ false を満たす $v \in \bigcup_i \mathrm{boundary}(S[i])$ の中で $C[v]$ が最小のもの;
5: $D[v_{\min}] \leftarrow$ true;
6: **for** $j \in \{i \mid v_{\min} \in \mathrm{boundary}(S[i])\}$ **do**
7: $\forall v \in V(S[j]), T[v] \leftarrow \infty$;
8: $\forall v \in \mathrm{boundary}(S[j])$, $T[v] \leftarrow C[v]$;
9: Dijkstra_distance$(S[j], T)$;
10: $\forall v \in \mathrm{boundary}(S[j])$, $C[v] \leftarrow T[v]$;
11: **end for**
12: **end for**
13: return C;

正しさの証明

このアルゴリズムの正しさは Dijkstra 法と同様に証明できる．境界頂点の中で，始点からの最短距離が 1 番目から $(i+1)$ 番目に短い頂点をそれぞれ $v_1, \ldots, v_{i+1}$ とおく．最短経路の性質より明らかに，始点から v_{i+1} への最短経路は，$v_1, \ldots, v_i$ のうちのいずれかの頂点 v_{prev} への最短経路に，v_{prev} と v_{i+1} の両方を含む小格子グラフだけを通るような v_{prev} から v_{i+1} への最短経路を付け加えたものである．よって，擬似コードの j 回目のループにおいて $(j = 1, \ldots, i)$，$v_{\min}$ に v_j が選ばれ，5 行目に到達した時点で $C[v_{\min}] = d(s, v_{\min})$ となっていたならば，$(i+1)$ 回目のループの頭に到達した時点で，$C[v_{i+1}]$ には $d(s, v_{i+1})$ が保持されており，5 行目では $v_{\min}$ に v_{i+1} が選ばれる．

計算量

このアルゴリズムの空間計算量は Asano らのアルゴリズムと同じく $O(k\sqrt{n} + n/k^2)$ であり，$k = k^*$ のときには $O(n^{2/3})$ となる．また，本アルゴリズムのループの各段階と多始点 Dijkstra 法のそれぞれにおいて，最小要素の探索を力任せに行う場合，このアルゴリズムの時間計算量は $O(k\sqrt{n} \times (k\sqrt{n} + n^2/k^4))$ である．$k = k^*$ のときは第二項が支配的になり，時間計算量は $O(n^2)$ となる．

3.2 Asano らの再帰アルゴリズムおよび分割方法の改善

入力された格子グラフを $G = (V, E)$，始点を $s \in V$ とおく．Asano らの再帰アルゴリズムにおいて，格子グラフの分割はパラメータ $k, l \in \mathcal{N}$ によって図 2 のように行う．まず G を k^2 個の小格子グラフに分割し，これらを 1 段目の小格子グラフと呼ぶ．再帰アルゴリズムでは，1 段目の各小格子グラフに対する走査の中で，小格子グラフ（図 2 では右上のグラフ $S_1[2]$）に多始点版 Dijkstra 法を走らせる代わりに小格子グラフを更に分割して 2 段目の小格子グラフ群（図では $S_2[1], \ldots, S_2[4]$）を作り，再帰的に最短距離を求めるアルゴリズムを適用する．同様にして分割と最短距離アルゴリズムの適用を l 回繰り返すと，1 段目から l 段目までの格子グラフがそれぞれ k^2 個ずつ作られる．l 段目の各小格子グラフに対して，2.1 節のアルゴリズムと同様に境界頂点の個数と同じ回数だけ走査を繰り返すと，l 段目の元になった $(l-1)$ 段目のある小格子グラフに対して多始点版 Dijkstra 法を走らせた場合と同じ結果が得られる．また，この操作を $(l-1)$ 段

目の各格子グラフについて行うと，$(l-1)$ 段目の各小格子グラフに対する一回の走査に相当する結果が得られる．同様に，これらの処理を再帰的に行うことで 1 段目の各境界頂点までの s からの最短距離が求まる．

頂点数 N_i の格子グラフを k^2 個に分割して作られた $(i+1)$ 段目の小格子グラフ群に関して，各小格子グラフには $O(N_i/k^2)$ 個の頂点が存在し，$O(\sqrt{N_i}/k)$ 個の境界頂点が含まれる．また，元の格子グラフ全体での境界頂点の数は $O(k\sqrt{N_i})$ となる．頂点数 N_i の格子グラフに対する最短距離計算の時間計算量を T_i とおくと，$i<l$ について $T_i = O((k\sqrt{N_i}) \times k^2 \times T_{i+1})$ という漸化式が成り立つ．アルゴリズム全体の時間計算量 $T = T_0$ は，$T_l = O((n/k^{2l})^2)$ より

$$T = T_0 = O((\frac{n}{k^{2l}})^2 \times \prod_{i=1}^{l}(k^{2-i}\sqrt{n} \times k^2)) \quad (1)$$

となり，これをまとめると

$$\begin{aligned} \frac{n^2}{k^{4l}} \times k^{4l-\sum_{i=1}^{l} i} \times (\sqrt{n})^l &= n^{2+\frac{1}{2}} \times k^{4l-\frac{l(l+1)}{2}-4l} \\ &= n^{2+\frac{1}{2}}/k^{\frac{l(l+1)}{2}} \quad (2) \end{aligned}$$

となる．また，多始点版 Dijkstra 法のために使用する領域と，再帰の各段で境界頂点への最短距離を保存するための領域が必要となるため，このアルゴリズムの空間計算量は

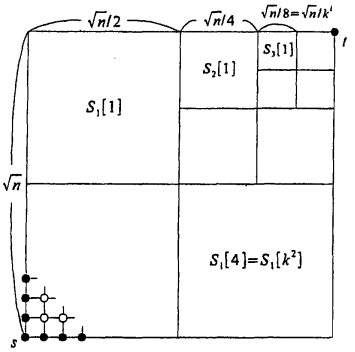

図 2: $k=2, l=3$ のときの格子グラフの分割例

$$O(k\sqrt{n} + \sqrt{n} + \frac{\sqrt{n}}{k} + \ldots + \frac{\sqrt{n}}{k^{l-2}} + \frac{n}{k^{2l}}) \quad (3)$$

となる．(3) 式は，例えば

$$(3) \leq k\sqrt{n} + 2\sqrt{n} + \frac{n}{k^{2l}} \quad (4)$$

で上界が与えられ，最適な分割数は $\Theta(n^{\frac{1}{2(2l+1)}})$ となる．このときの空間計算量は $O(n^{\frac{l+1}{2l+1}})$ であり，時間計算量は $O(n^{2+\frac{l}{2}-\frac{l(l+1)}{4(2l+1)}})$ となる．

Asano らのアルゴリズムでは全ての段で同じ分割数を用いているが，それぞれの段において頂点数に応じた適切な分割数を設定することで，同じ段数 l に対して使用メモリをさらに削減できる．言い換えれば，同じ空間計算量を達成するために必要な段数が少なくなるため，時間計算量が改善される．

いま，i 段目の分割数を k_i，i 段目から l 段目までの合計の空間計算量を S_i とおくと，各 $i=1,\ldots,l$ について $N_{i-1} = O(k_i^2 N_i)$，$S_{i-1} = O(k_i\sqrt{N_i} + S_i)$ となる．ただし，N_0 は入力された格子グラフ G の頂点数 n，S_0 はアルゴリズム全体の空間計算量 S である．

ここで，数列 a_i を用いて $S_i = O(N_i^{a_i})$ と表すと，$i \leq l$ に関して

$$S_{i-1} = O(k_i\sqrt{N_{i-1}} + (N_{i-1}/k_i^2)^{a_i}) \quad (5)$$

と表せる（S_l は $O(N_l)$）．これを k_i で微分することで，S_{i-1} を最小化する k_i^* とそのときの S_{i-1} は

$$k_i^* = \Theta(N_{i-1}^{\frac{2a_i-1}{2(2a_i+1)}}), \quad (6)$$

$$S_{i-1} = O(N_{i-1}^{\frac{2a_i}{2a_i+1}}) \quad (7)$$

となる．以上より得られた a_i の漸化式を解く[2]と $a_i = 2^{l-i}/(2^{l-i+1}-1)$．よって，使用メモリが最小となるような分割を行う場合の i 段目の分割数 k_i^* は N_{i-1} を用いて

$$k_i^* = \Theta(N_{i-1}^{\frac{1}{2(2^{l-i+2}-1)}}) \quad (8)$$

[2] $a_i = 1/A_i$ とおくと $A_{i-1} = 1 + \frac{1}{2}A_i$ という漸化式が得られるため，これを解けばよい．

となる．このときのアルゴリズム全体の空間計算量 S は

$$S = S_0 = O(N_0^{a_0}) = O(n^{\frac{2^l}{2^{l+1}-1}}) \tag{9}$$

となる．

次に，同様にして時間計算量を求める．i 段目の小格子グラフに対する時間計算量を T_i とおくと，各 $i = 1, \ldots, l$ に対して $T_{i-1} = O((k_i^2\sqrt{N_i}) \times k_i^2 \times T_i)$．ただし，$T_0$ はアルゴリズム全体の時間計算量 T である．ここで，数列 b_i を用いて $T_i = O(N_i^{b_i})$ と表すと，$i \leq l$ に関して

$$T_{i-1} = O(k_i\sqrt{N_{i-1}} \times k_i^2 \times (N_{i-1}/k_i^2)^{b_i}) \tag{10}$$

となる（T_l は $O(N_l^2)$）．これに $k = k^*$ を代入して得られた b_i の漸化式を解く[3]と $b_i = \{2^{l-i}(l-i+3) - 1\}/(2^{l-i+1}-1)$．よって，使用メモリが最小となるような分割を行う場合のアルゴリズム全体の時間計算量 T は

$$T = T_0 = O(N_0^{b_0}) = O(n^{\frac{2^l(l+3)-1}{2^{l+1}-1}}) \tag{11}$$

となる．

3.3 二つの改善を合わせたアルゴリズムの性能評価

本節では，3.2 節の分割手法を用いた再帰アルゴリズムの各段に対する計算を 3.1 節の高速アルゴリズムで置き換えた場合の時間計算量を考える．

i 段目の小格子グラフに対する時間計算量を T_i とおくと，分割の段数が l であるとき，各 $i = 1, \ldots, l$ に対して $T_{i-1} = O(k_i^2\sqrt{N_i} \times (k_i^2\sqrt{N_i} + T_i))$ となる．括弧内の第一項は値が未確定の最小要素を力任せに探すための時間である．3.1 節の議論などにより，$T_l = O(N_l^2)$, $T_{l-1} = O(N_{l-1}^2)$．ここで，数列 c_i を用いて $T_i = O(N_i^{c_i})$ と表すと，

$$T_{i-1} = O(k_i\sqrt{N_{i-1}}(k_i\sqrt{N_{i-1}} + (N_{i-1}/k_i^2)^{c_i})) \tag{12}$$

となる．

このアルゴリズムの空間計算量は明らかに前節のアルゴリズムと等しいことから，$k_i = k_i^*$ のときに空間計算量は最適化される．そして，上式に $k_i = k_i^*$ を代入して第一項と第二項を比較すると，$c_i \geq 2^{l-i}/(2^{l-i+1}-1)$ のとき第二項が支配的になることが分かる．c_i は明らかに 1 よりも常に大きく，また $2^{l-i}/(2^{l-i+1}-1) \leq 1$ であるため，$k_i = k_i^*$ のとき

$$T_{i-1} = O(k_i^*\sqrt{N_{i-1}} \times (N_{i-1}/k_i^{*2})^{c_i}) \tag{13}$$

となる．

得られた c_i の漸化式を解く[4]と $c_i = 2^{l-i}(l-i+2)/(2^{l-i+1}-1)$．よって，使用メモリが最小となるような分割を行う場合のアルゴリズム全体の時間計算量 T は

$$T = T_0 = O(N_0^{c_0}) = O(n^{\frac{2^l(l+2)}{2^{l+1}-1}}) \tag{14}$$

となる．式 (11) を変形すると，

$$T = O(n^{\frac{2^l(l+2)}{2^{l+1}-1} + \frac{2^l-1}{2^{l+1}-1}}) \tag{15}$$

となることから，このアルゴリズムは再帰の分割方法を変えただけのアルゴリズムよりも $O(n^{\frac{2^l-1}{2^{l+1}-1}})$ 倍高速である．

図 3 は，Asano らのアルゴリズムと我々の提案アルゴリズムの計算量とパラメータの関係を表す．

[3] $b_i = 2^{l-i}B_i/(2^{l-i+1}-1)$ とおくと $B_{i-1} = B_i + 1 + 1/2^{l-i+1}$ という漸化式が得られるためこれを解けばよい．
[4] $c_i = 2^{l-i}C_i/(2^{l-i+1}-1)$ とおくと $C_{i-1} = C_i + 1$ という漸化式が得られるためこれを解けばよい．

AD は Asano らの最短距離を求める再帰アルゴリズム，3.2 は 3.2 節の分割数だけを改善したアルゴリズム，3.3 は本節のアルゴリズムを表している．横軸は空間計算量，縦軸は時間計算量を表し，図中の円はそれぞれ右下から $l = 1, 2, 3, \ldots$ のときの計算量を表す．

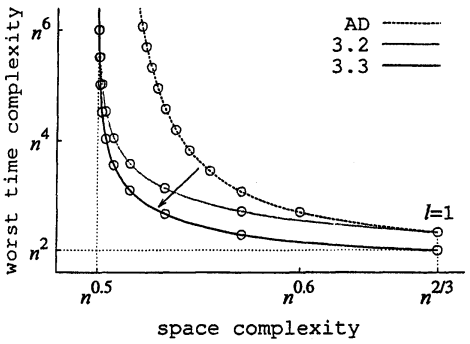

図 3: 計算量とパラメータの関係

3.4 最短経路を出力する再帰アルゴリズム

最後に，最短距離を求めるのと同様の方法を用いて，最短経路を出力するアルゴリズムを改善する方法を示す．

2.2 節の Algorithm2 において，AD_path は Dijkstra と同等の動作をするため，各小格子グラフに対して Dijkstra を用いる代わりに AD_path を再帰的に用いることができる．また，最短距離を求める場合と同様に，再帰の各段ごとに分割を変化させ，最短距離計算では我々の提案アルゴリズムを用いることによって，より効率の良い時間・空間計算量で計算を行うことができる．更に，Algoritm 2 では始点から各境界頂点への最短距離を求めるのに AD_distance を繰り返して用いているが，これは一度だけ実行すれば十分である．

このアルゴリズム全体の空間計算量は，3.2 節と同様に適切な分割数をとることにより $O(n^{\frac{2^l}{2^{l+1}-1}})$ に最小化される．また，i 段目の小格子グラフに対する我々の再帰的最短距離アルゴリズムの時間計算量を TD_i，最短経路アルゴリズムの時間計算量を TP_i，頂点数を N_i，分割数を k_i とすると，分割の段数が l であるとき，各 $i = 1, \ldots, l$ について

$$TP_{i-1} = O(k_i^2\sqrt{N_i} \times (TD_i + TP_i)) \tag{16}$$

となる．ただし，TD_0, TP_0 は各アルゴリズムの全体の時間計算量 TD, TP である．

ここで，$TD_{i-1} = O(k_i^2\sqrt{N_i} \times TD_i)$ であり，TP_l と TD_l がともに $O(N_l^2)$ であることから，TD_i と TP_i はオーダの意味で等しくなる．よって，使用メモリが最小となるような分割を行う場合のアルゴリズム全体の時間計算量 TP は TD と同じく $TP = O(n^{\frac{2^l(l+2)}{2^{l+1}-1}})$ となる．

4 結論と今後の課題

本研究では，格子グラフ上の二頂点間の最短距離を求めるための，既存手法よりも効率の良いアルゴリズムを提案した．

今後の課題の一つとしては，提案アルゴリズムをより一般的な問題，例えば平面グラフ上の最短経路問題など，に拡張することが考えられる．

参考文献

[1] Asano, T. and Doerr, B., "Memory-Constrained Algorithms for Shortest Path Problems", Proceedings of 23rd CCCG, 2011.

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