Vol.2015-AL-152 No.8
2015/3/3

重みつき障害物を含む平面上での最短経路アルゴリズム

早川 裕真[1]　浅野 哲夫[1]

概要：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上で解くためには，グラフの大きさに比例した多くのメモリが必要になる．本研究では，平面上の最短経路問題を解く実用的でメモリ効率の良いアルゴリズムの開発を行う．また，現場では障害物をある対価を払って通過することがあり，そのような障害物に重みがついている場合への拡張も行う．

キーワード：アルゴリズム，最短経路問題，重みつき障害物

Hayakawa Yuma[1]　Asano Tetsuo[1]

Abstract: Recently, finding a shortest path in a big graph is required as society becomes complex. However, it requires huge memory proportional to the size of the graph when we use a computer to solve it. In this paper, we propose an efficient and practical memory constraint algorithm that solves the shortest path problem on a plane. It can be extended to the case that when obstacle has its cost, and a field robot can pass it with paying the cost.

Keywords: algorithm, shortest path problem, obstacle with weight

1. はじめに

2011 年 3 月 11 日に東日本大震災と巨大津波により，福島第一原子力発電所はこれまで経験したことのない原子力災害を引き起こした．人間に代わって高濃度に汚染された原子炉建屋での作業をロボットに求められたが，バッテリーの問題でロボットの活動時間に制限がある中で作業を行わなければならず，ロボットが目的地まで最短経路で到達することの重要性が強く認識された．イメージ図を**図 1** に示す．

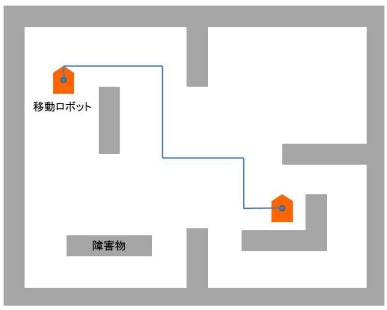

図 1　ロボットの経路計画

[1] 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

計算幾何学における問題の一つとして障害物を含む平面上の 2 点間の最短経路を求める問題があり，Hershberger と Suri[**1**] は $O(n \log n)$ 時間で計算するアルゴリズムを提案した．障害物と平面上の 2 点がある図を**図 2** に示す．

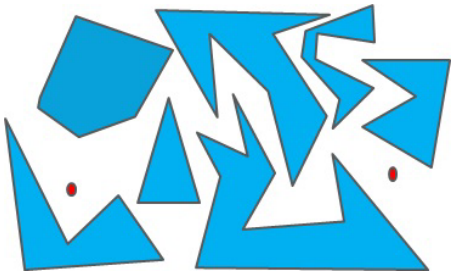

図 2　障害物と平面上の 2 点

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上で解くためには，グラフの大きさに比例した多くのメモリが必要になる．Asano と Doerr[**2**] は格子グラフ上での最短経路アルゴリズムを提案した．これは，重みつきの辺と s と t の 2 つの頂点を含むサイズ $\sqrt{n} \times \sqrt{n}$ の格子グラフが与えられたとき，$O(n^{\frac{1}{2}+\varepsilon})$ の作業領域だけを用いて多項式時間で動作するアルゴリズムである．ただし，

ε は任意の正の実数である．このアルゴリズムは格子グラフを k^2 個の小格子グラフに分割し，小格子グラフの境界点だけで距離情報を管理することで実現している．

1.1 研究目的

重みがついた領域上の最短経路を平面分割により求める研究として [3] がある．これは，重みつきの多角形と始点 s と終点 t が与えられたとき，$O(n^4)$ の作業領域を用いると $O(n^8)$ の計算時間で最短経路を求めるアルゴリズムである．ここで，n は障害物の頂点の総数を表す．しかし，格子グラフを作成しないこの手法は複雑で計算時間が長い．そこで本研究では，平面上の最短経路問題を解く実用的でメモリ効率の良いアルゴリズムの開発を行う．また，現場では障害物をある代償を払えば通過できることがあり，そのような障害物に重みがついている場合への拡張も行う．

本研究の特色は点位置決定アルゴリズムを利用して，一般の重みつき障害物の入力を格子グラフの形に変換することにある．省メモリのためには実際に格子グラフを求めるのではなく，必要に応じて辺の重みを計算で求めることが必要となる．

1.2 問題の定義

まず，本研究で考える最短経路問題を定義する．平面は直線によって領域に分かれていると仮定する．障害物を通過する際に払う代償を考慮するため，領域ごとにそれぞれ重みをつける．平面上には始点 s, 終点 t の 2 点があり，s から t への最小コストのものを見つける問題を本研究で考える最短経路問題と定義する．コストとは距離と重みの積とする．問題の図を**図 3** に示す．

また，本研究では経路上を移動する物体を点と仮定する．なぜなら実際のロボットを移動対象にすると，回転時の角度など複雑な制約が増え，アルゴリズムが複雑になるためである．しかし，本研究で用いるアルゴリズムは障害物などの幅をロボットの幅の分だけ太らせることで，回転などは考慮出来なくても，ロボットの幅を考慮した経路を出力することができる．

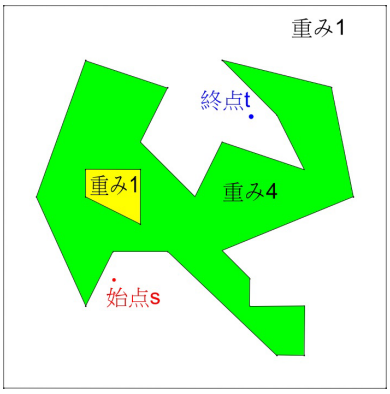

図 3 問題の図

2. 格子グラフ上での最短経路アルゴリズム

本節では，重みつき障害物を含む平面上での最短経路問題を格子グラフをあてはめて近似的に解くアルゴリズムについて説明する．

2.1 本研究のアルゴリズムの概要

重みつき障害物を含む平面上での最短経路問題を近似的に解く手順は以下のようになる．

本研究では次に示す順序で処理を行い，近似的に最短経路を発見する．

(1) 格子グラフを作成する．

(2) 格子の頂点がどの多角形の内部に属するか判定し，頂点の重みを計算する．

(3) 頂点の重みから辺の重みを計算する．

(4) ダイクストラ法で最短距離を求める．

ダイクストラ法とは，最短経路問題を効率的に解くグラフ理論におけるアルゴリズムであり，手近で明らかなところから距離を順次確定していき，その確定した情報をもとにさらに遠くまで確定していく方法である．

各手順の詳しい説明を以下に示す．

2.2 格子グラフの作成

本研究では，地図上に縦と横，斜めの直線を等間隔で任意の本数分作成する．斜めの直線とは，斜め 45° の向きの直線である．各直線により交点ができるが，その交点を頂点とし，頂点と頂点を結ぶものを辺とする．このようにしてできたグラフを本研究では格子グラフと呼ぶ．作成した格子グラフを**図 4** に示す．この格子グラフの頂点を始点や終点とし，通る辺を経路とする．これにより，簡易に平面上を通る経路を出力することができる．その様子を**図 5** に示す．しかし本研究のアルゴリズムで得られる経路は，格子上を通る経路のためマンハッタン距離 (L1 距離) となる．例えば**図 6** のように格子上に始点と終点があるとき，それを結ぶ経路は数種類あるが，距離は全て同じである．よって，実用的な経路とは違う．そのための補正の第一歩として，斜め 45° の向きにも進めるようにしたが，本質的には解決していない．

マンハッタン距離となってしまうことにより，実用的でない経路になる問題を少しでも解決するために，同じ距離の経路になる場合，向きを変える回数が多い経路を選択するようにした．例えば**図 6** の場合，② のコースのような経路である．その結果，格子の本数を増やした時，② の

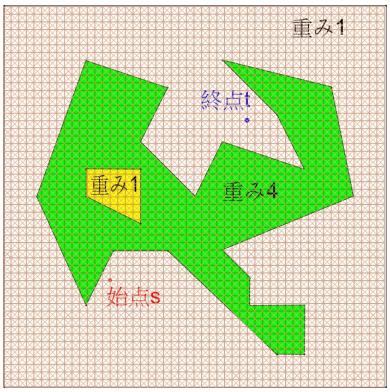

図 4　格子グラフ作成

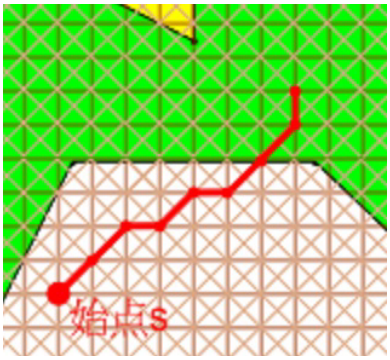

図 5　経路を辿る様子

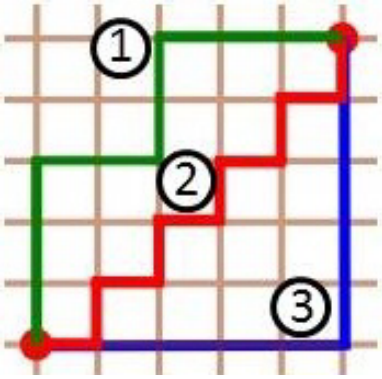

図 6　マンハッタン距離

コースのような経路になるので，視覚的には点と点を最短距離で結んでいるようにみえる．そのアルゴリズムを **Algorithm1** に示す．

Algorithm 1 マンハッタン距離になる場合，向きを変える回数が多い経路を選択するアルゴリズム

```
1: //Backtracking
2: if 候補となる経路が複数 then
3:     if 連続で同じ向きの経路 then
4:         違う向きの経路を探索する
5:     end if
6: end if
```

2.3　辺の重みの計算

本研究では隣接するノード（頂点）に行くコスト（辺の重み）を頂点の値を用いて求める．これにより障害物の重みを考慮した経路が実現している．計算式は **(1)** 式，**(2)** 式となる．

- 2 点のノードが上下左右に位置している場合

$$\text{2 点間の辺の重み（コスト）} = \frac{\text{自分の頂点の領域の重み} + \text{相手の頂点の領域の重み}}{2} \tag{1}$$

- 2 点のノードが斜めに位置している場合

$$\text{2 点間の辺の重み（コスト）} = \frac{\text{自分の頂点の領域の重み} + \text{相手の頂点の領域の重み}}{2} \times 1.5 \tag{2}$$

(2) 式の末尾は本来は $\times\sqrt{2}$ であるが，今回は便宜上簡略化し $\times 1.5$ とした．

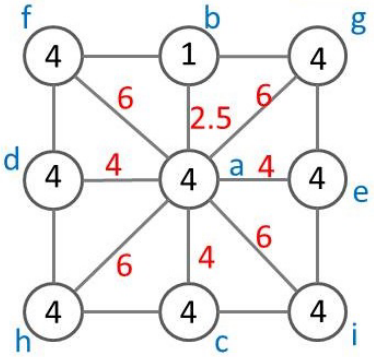

図 7　頂点の重みと辺の重み

例えば**図 7** の a と b を結ぶ辺の重み（コスト）は

$$\frac{4+1}{2} = 2.5 \tag{3}$$

となり，**図 7** の a と f を結ぶ辺の重み（コスト）は

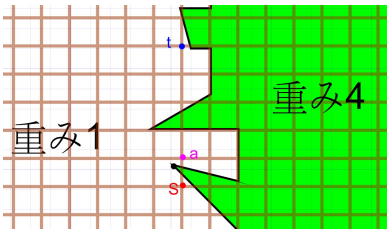

図 8　辺の重みの近似

$$\frac{4+4}{2} \times 1.5 = 6 \qquad (4)$$

となる．

しかし，図 8 のような状況で s 点から a 点を経由して t 点に行く経路になった場合，s 点と a 点間の辺の重みは 1 となる．そのため，本手法では角度が小さい鋭角をもつ多角形が与えられたとき，場合によっては正確な辺の重みが求められない．

2.4　小格子グラフに分割して解くアルゴリズム

ここではアルゴリズムを省メモリ化するために，[2] のアルゴリズムを導入する．

2.4.1　最短距離の計算

[2] のアルゴリズムはまず作成した格子グラフを k^2 個の小格子グラフに分割する．各小格子グラフを S_{00}，S_{01}，...S_{0k-1},S_{10}，S_{11}，...S_{k-1k-1} といったように番号をつける．S の添え字の 10 の位の数が行番号で 1 の位の数を表している．各小格子グラフの周りを囲む点を区別して境界点と呼ぶ．境界点は二つの小格子グラフに属している点が存在する．（三つ，四つの小格子グラフに属している点もある．）

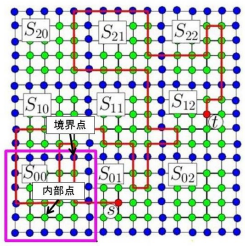

図 9　最短距離の計算手順 1

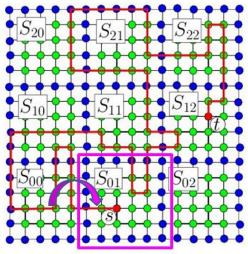

図 10　最短距離の計算手順 2

図 9 のように小格子グラフに分割後，各小格子グラフに属する頂点のみに対しダイクストラ法を実行し，確定した点からの最短距離を随時確定していく．つまり，最初確定している点は始点のみのため，始点が存在する小格子グラフに達するまで各頂点には初期値が入力されたままである．各小格子グラフに属するすべての頂点までの暫定最短距離がダイクストラ法によって計算された後，境界点のみの暫定最短距離を記憶しておく．境界点以外の内部の点は忘れる．そして，次の小格子グラフに対し，同様にダイクストラ法を実行する．二つの小格子グラフに属している境界点には片方の小格子グラフに対してのダイクストラ法を実行したことによって計算され，確定した暫定最短距離の値が入力されている．そして図 10 のようにこの小格子グラフに対してのダイクストラ法を分割して出来た小格子グラフ全てに対して行う．これをスキャンと呼ぶことにする．そしてこのスキャンを境界点の総数分繰り返す．なぜなら，図 11 のように小格子グラフを跨いで一度ダイクストラ法を実行した小格子グラフに戻ってくる経路になる場合があるからだ．戻る回数は高々境界点の総数分になる．一連の手順を **Algorithm2** に示す．

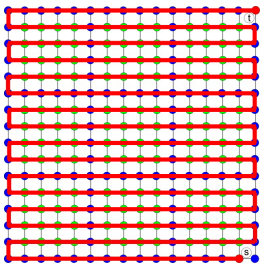

図 11　境界点の総数分繰り返す

各小格子グラフには頂点が $O(\frac{\sqrt{n}}{k} \times \frac{\sqrt{n}}{k}) = O(\frac{n}{k^2})$ 個ある．ダイクストラ法の時間計算量は $O(n^2)$ なので各小格子グラフにおいての時間計算量は $O(\frac{n^2}{k^4})$ である．小格子グラフは全部で k^2 個あるので，1 回のスキャンの時間計算量は $O(\frac{n^2}{k^4} \times k^2) = O(\frac{n^2}{k^2})$ である．このスキャンの回数は高々境界点の総数分となる．境界点の総数は $O(\frac{\sqrt{n}}{k} \times k^2) = O(k\sqrt{n})$ となるので，小格子グラフに分割して最短距離を求める時間計算量は $O(\frac{n^2}{k^2} \times k\sqrt{n}) = O(\frac{n^{2+\frac{1}{2}}}{k})$ である．

作業領域は小格子グラフ内でダイクストラ法を実行する際の小格子グラフの頂点全ての距離情報と他の小格子に移ったあと始点から境界点までの距離情報が必要となるので，全体の作業領域は $O(\frac{n}{k^2} + k\sqrt{n})$ である．

また $O(\frac{n}{k^2} + k\sqrt{n})$ の値が最も小さくなるときは $k = n^{\frac{1}{6}}$ のときで，そのときの作業領域は $O(n^{\frac{2}{3}})$ である．

2.4.2　最短経路の計算

内部点には最短距離値は記憶されておらず，境界点には最短距離計算時に求めた始点からの最短距離が記憶されている．そのため，まずは最短距離の計算後，図 12 のように終点が属している小格子グラフに対して，ダイクストラ法を実行し，終点からすべての境界点までの最短距離を計算する．そしてその小格子グラフのすべての境界点に対し，以下の **(5)** 式で確かめる．

$$D(s, v_i) + d(v_i, t) = d(s, t) \qquad (5)$$

始点から各境界点頂点までの距離を $D(s, v_i)$，ダイクストラ法によって求めた各境界点から終点までの距離を $d(v_i, t)$，最短距離を計算するアルゴリズムで求めた始点から終点ま

での距離を $d(s,t)$ とする．

(5) 式でイコールになる境界点が最短距離値を出力する経由地になっていることがわかる．イコールとなる境界点が複数あった場合，その中で一番小さい値をもつ境界点を選ぶ．そして今度は**図 13** のように境界点が属しているもう片方の小格子グラフに対して，ダイクストラ法を実行し，その境界点からすべての境界点までの最短距離を計算する．

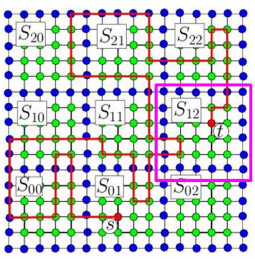

図 12 最短経路の計算手順 1

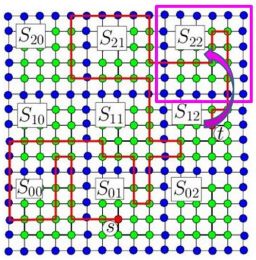

図 13 最短経路の計算手順 2

これを始点まで繰り返すことにより最短経路を求める．最短経路を計算する手順を **Algorithm3** に示す．

時間計算量は，最短距離を計算するアルゴリズムを高々境界点の総数分の $k\sqrt{n}$ 繰り返すこととなるので $O(\frac{n^{2+\frac{1}{2}}}{k} \times k\sqrt{n}) = O(n^3)$ である．作業領域は最短距離を計算するアルゴリズムと同じなので $O(\frac{n}{k^2} + k\sqrt{n})$ である．

3. 実験：格子上で求める手法の検証

このアルゴリズムが実用的なアルゴリズムかどうかを検証するため，計算機で実験を行った．言語は C 言語を使用し，ライブラリは LEDA を使用する．LEDA とは，独マックスプランク研究所で開発されたオブジェクト指向の C++クラスライブラリである．『Library of Efficient Data types and Algorithms（効率的なデータ型とアルゴリズムのライブラリ）』の頭文字をとってその名が付けられた．

Algorithm 2 格子グラフ上の 2 点間の最短距離を計算するアルゴリズム

Input: 辺の重みをもった $O(\sqrt{n}) \times O(\sqrt{n})$ のサイズの格子グラフ G，$\sqrt{n}$ は整数，小格子グラフの分割数 k，始点 s，終点 t
Output: 始点 s から終点 t への最短距離
//始点から各境界点まで距離を配列 C[] と定義する．
//ある小格子グラフの各頂点においての距離を配列 T[] と定義する．

for グラフ G の全ての境界点 **do**
  $C[i][j] = \infty$
end for
C[s]=0
T[s]=0
for *round* 1 *to* $k\sqrt{n}$ **do**
  for 各小格子グラフ S_{ij} 全て **do**
    for S_{ij} の中の境界点 **do**
      $T[i][j] = C[i][j]$
    end for
    for S_{ij} の中の内部点 **do**
      $T[i][j] = \infty$
    end for
    //ダイクストラアルゴリズム
    while S_{ij} の内部の非選択の頂点がある **do**
      未確定の頂点から T[i][j] が最も小さい頂点を選ぶ
      選択した点に印をつける
      if 未確定の頂点 $T[q]$ > 確定している頂点 $T[p]$+ 辺の重み w **then**
        $T[q] = T[p] + w$
      end if
    end while
    //配列 C へ結果を移す
    for S_{ij} の中の境界点 **do**
      $C[i][j] = T[i][j]$
    end for
  end for
end for
if 終点が境界点 **then**
  return 始点から終点までの最短距離 C[t]
end if
if 終点が内部点 **then**
  return 始点から終点までの最短距離 T[t]
end if

Algorithm 3 格子グラフ上の 2 点間の最短経路を計算するアルゴリズム

Input: 辺の重みをもった $O(\sqrt{n}) \times O(\sqrt{n})$ のサイズの格子グラフ G，$\sqrt{n}$ は整数，小格子グラフの分割数 k，始点 s，終点 t，始点から終点までの最短距離値，始点 s から各境界点までの最短距離値 C[i][j]
Output: 始点 s から終点 t への最短経路
//始点から各境界点まで距離を配列 C[] と定義する．
//ある小格子グラフの各頂点においての距離を配列 T[] と定義する．

1: 経路を描いた点までの最短距離値 leng=始点から終点までの最短距離値
2: **while** $leng > 0$//始点に到達するまで繰り返す **do**
3: T[t]=0
4: 終点が属する小格子グラフの番号を求める
5: //ダイクストラアルゴリズム
6: **while** S_{ij}の内部の非選択の頂点がある **do**
7: 未確定の頂点から T[i][j] が最も小さい頂点を選ぶ
8: 選択した点に印をつける
9: **if** 未確定の頂点 $T[q] >$ 確定している頂点 $T[p]+$ 辺の重み w **then**
10: $T[q] = T[p] + w$
11: **end if**
12: **end while**
13: **for** S_{ij}の中の境界点 **do**
14: **if** $C[i][j] + T[i][j] == leng$ **then**
15: T[i][j] が最小な頂点を選ぶ
16: **end if**
17: **end for**
18: //Backtracking
19: **if** leng 値が格納されている点の周囲の点 $T[p] + w == leng$ **then**
20: w の向きに経路を描く
21: $leng = leng - w$
22: **end if**
23: **if** 次は選んだ T[i][j] が最小な頂点 **then**
24: 最後に描いた点が属するもう片方の小格子グラフの番号を求める
25: 上の指示（ダイクストラアルゴリズム）に戻る
26: **end if**
27: **end while**

3.1 実験 1：格子の本数と経路

格子の本数を増やせば増やすほど，実用的な経路を出力することができるが，その分メモリは増大する．そこで実用的な経路を出力することができる格子の本数を調査する．

3.1.1 実験環境

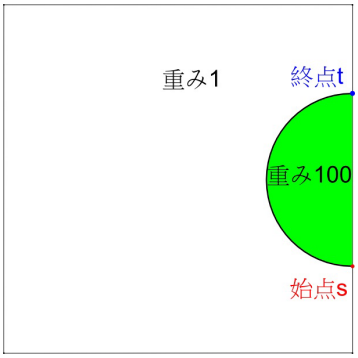

図 14 実験 1 の実験環境

図 14 のように，始点と終点の 2 点，半円を右端に配置し，重みのことなる領域を二つ設定した．半円の外側の領域の重みが 1，半円で囲まれた領域の重みが 100 とする．そして**図 14** の上にひく格子の本数を変化させる．半円の内部を通る経路にならないようにし，円周を辿るような経路を出力させて，曲線に近い格子の本数を調査する．

3.1.2 結果

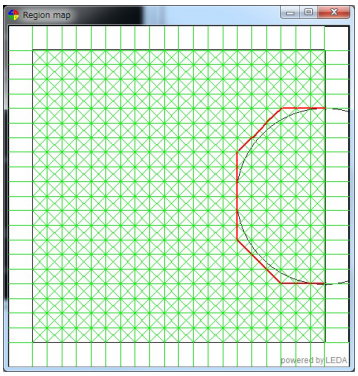

図 15 格子 20 本

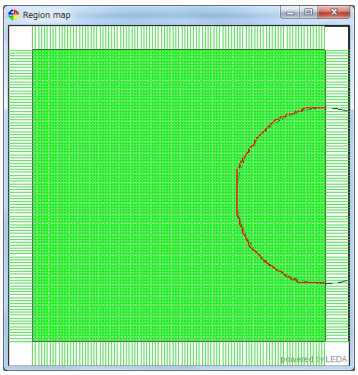

図 16 格子 100 本

格子 20 本のときの経路を**図 15** に，格子 100 本のときの経路を**図 16** に示す．格子 20 本（頂点数：400）のときは滑らかでない経路になっている．格子 100 本（頂点数：10000）のときは視覚的には滑らかな曲線の経路になっている．また半円の中心からすべての経路上の頂点までのユークリッド距離を計算し，その中で最大となる距離を求めた．そしてそこから理想の値（半円の半径 (180.0)）を引き，理想の値との差を求めた．平面上の経路ならば理想の値との差が 0 に近い値になる必要がある．グラフにしたものを**図 17** に示す．

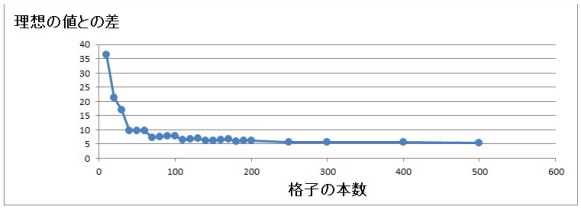

図 17　本数と理想の値との差

本数を増やしていくと差が縮まるのがわかる．よって格子の本数を増やすほど，実際な経路により近くなる．また本数を 500 本（頂点数：250000）と増やしても差が 0 に近い値にならなかった．これは格子の本数を増やし，頂点の数が多くなっても，曲線で滑らかな実際の経路を出力することは困難であることを示している．

また本研究のアルゴリズムでは，多く格子をひけばより円周を沿う経路になり滑らかな経路になるが，メモリ数が増大し，計算時間も長くなる．例えば格子の本数が 10 倍になると作業領域は約 20 倍，100 倍の本数になると作業領域は約 450 倍となる．また計算時間は格子の本数が 10 倍になると約 10 万倍，本数が 100 倍になると約 100 億倍となる．よって 100 本の格子である程度の滑らかな曲線になっているため，以降の実験では格子の本数を 100 本とする．

3.2　実験 2：重みと経路の関係と本手法の分析

重みを考慮した最短経路にどの程度近い値になっているか実験的に確かめる．また本手法によって得られた経路を分析する．

3.2.1　実験環境

図 3 のように，始点と終点の 2 点，重みのことなる領域を三つつ設定した．中央の大きい多角形に囲まれて，中央よりやや左に位置する小さい四角形で囲まれた領域には含まれない領域の重みを 4 とし，その外側の領域を重み 1 とする．小さい四角形に囲まれた領域の重みを 1〜100 まで変化させる．

3.2.2　結果

経路は 2 種類に分かれた．ここでは 1 つ目の経路を代表して重み 1 のとき，2 つ目の経路を代表して重み 4 のときの経路とする．小さい四角形で囲まれた領域が重み 1 のときの実行結果を**図 18** に示す．重みが小さい領域を長く通る経路となっている．また重み 4 の領域の部分はなるべく短い距離をとる経路となっている．

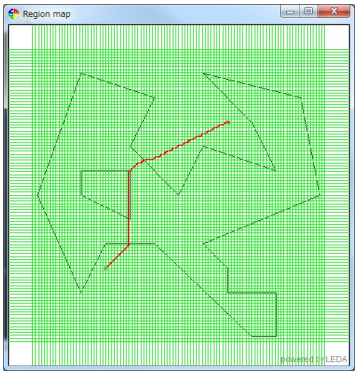

図 18　重み 1 のとき

小さい四角形で囲まれた領域が重み 4 のときの実行結果を**図 19** に示す．重みが 1 から 4 になったので，重み 1 の領域の部分は避けて，重み 4 の領域を通る経路となっている．重み 4 の領域を出たあとは，重み 4 の領域のすぐ外側で外枠の重み 1 の領域内を通る経路となっている．

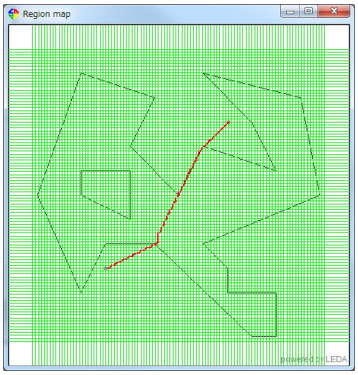

図 19　重み 4 のとき

また，さらに結果を分析をするため重み 1 と重み 4 のときのコストと経路の長さを比較した．その様子を**表 1** に示す．

表 1　実験 2 の数値結果

重み	コスト	経路の長さ	経路の長さ/コスト
1.0	120.0	464.132034	3.867766950
4.0	134.0	436.014285	3.253837948

重みが 1 の経路のコストは，重み 4 の経路のコストより低くなっている．しかし，経路の長さは重み 1 のときの方が重み 4 のときより長くなっている．これは重みが小さい領域を通る経路はコストが小さくて済むが経路が長くなり，重みが大きい領域を通る経路はコストは大きいが経路は短くて済むことを表している．また経路の長さをコスト

で割った 1 コストあたりの経路の長さは重み 1 のときの方が大きくなっている．よって重み 1 の経路は低いコストで長い距離を進む経路になっている．

時間計算量に関してはダイクストラ法の時間計算量が辺の数を m，頂点の数を n としたとき $O(m + n\log n)$ である．そのため格子の本数を $\sqrt{n}$ とした時（n 個の格子頂点数）$m \approx 2n$ となるので，それぞれダイクストラ法の時間計算量に代入すると時間計算量は $O(n\log n)$ となる．さらに小格子に分割して最短距離を計算するアルゴリズムを適用すると $O((n\log n)^{2+\frac{1}{2}})$ となる．作業領域はダイクストラ法の時間計算量と同じなので $O(n)$ となる．さらに小格子に分割して最短距離を計算するアルゴリズムを適用すると $O(n^{\frac{2}{3}})$ となる．

3.3 実験のまとめ

計算機実験により，考案したアルゴリズムは重みの値が考慮された経路を計算できるアルゴリズムであることがわかった．しかし，実験 1 において格子の本数を増やしても完全に滑らかな経路にはならなかった．さらに実験 2 の重み 4 のときの経路が，重み 4 の領域のすぐ外側で外枠の重み 1 の領域内から重み 4 の領域の方向へ近づくことと，重み 1 の領域の方向へ離れることを何度も繰り返す経路となっており，実際の経路とは少し異なっており，完全に一致する経路とはなっていない．

4. おわりに

4.1 まとめと今後の課題

今回は格子グラフを作成して，重みつき障害物を含む平面上での最短経路問題を求め，C 言語で実装した．そして，考案したアルゴリズムを検証するために計算機実験を行った．**[2]** のアルゴリズムを導入し，$O(n^{\frac{2}{3}})$ の作業領域で $O((n\log n)^{2+\frac{1}{2}})$ で動作するアルゴリズムとなった．また，本研究では複雑なアルゴリズムを用いずに最短経路を計算することを目標としたため，経路を計算することができたが格子上の経路であって実際の平面上の経路とはなっていない．そのため経路の誤差がどうしても生じる．本研究では実際の経路とは完全には一致しないが，実用的な経路を $O(n^{\frac{2}{3}})$ の少ない作業領域を用い $O((n\log n)^{2+\frac{1}{2}})$ の短い計算時間で計算することができた．今後は制度が高い平面上の経路が計算出来，かつ複雑なアルゴリズムにならない方法を見つけることが課題となる．

参考文献

[1] John Hershberger and Subhash Suri,” An optimal algorithm for euclidean shortest paths in the plane”，SIAM J.COMPUT，Volume 28，No.6，pp.2215-2256，1999

[2] Asano, T. and Doerr, B.,” Memory-Constrained Algorithms for Shortest Path Problems”，CCCG，2011

[3] Joseph S. B. Mitchell and Christos H. Papadimitriou,” The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision”，JACM，Volume 38 Issue 1，pp．18-73，Jan．1991