修 士 論 文

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

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

早川 裕真

2015 年 3 月

修 士 論 文

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

指導教員　上原隆平 教授

審査委員主査　上原隆平 教授
審査委員　平石邦彦 教授
審査委員　緒方和博 教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

1310057 早川 裕真

提出年月: 2015 年 2 月

概 要

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．高濃度に汚染された原子炉建屋での作業は人間に代わって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．また自動車である目的地に行く際に用いるカーナビは非常に便利で運転者の負担を軽減している．カーナビは現在位置から目的地までの最短経路を求め，出発前に運転者に提示している．さらに今日では人工知能で制御された自動車が運転者の操縦なしに走行する自動運転の開発が日々行われている．そのため，瞬時に最短経路を求める必要性がさらに増している．

最短経路を見つけることは障害物を含む平面上の 2 点間の最短経路を求める問題として計算幾何学で扱われている．この問題は平面上に多角形の障害物がいくつか与えれているとき，指定された始点と終点を結ぶ最短経路を見つける問題である．

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．Mitchell と Papadimitriou[**4**] 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が求めるアルゴリズムを提案した．ここで，N は障害物の頂点の総数を表す．しかし，この手法は複雑で計算時間が長い．本研究では，格子グラフを用いることで障害物を考慮した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うことで移動ロボットが通過することがあり，距離と重みを考慮した経路を選択しなければならない状況がある．例えば，出発点から目標点の直線上に池があり，池の近くには雑草がある環境で移動ロボットが走行する経路を考える．地図上に障害物がなければ最短経路は目標地点へ直線に進む経路になる．ここでは，移動ロボットは池を通過することが困難であると仮定すると，目標地点への直線上に池があるので直線に進む経路は最短経路にならない．それ以外で考えられる最短経路は，遠回りだが障害物がない経路と距離は短いが道の途中に雑草がある経路の 2 通りの経路であると仮定する．この場合の最短経路は雑草を通過する経路になる．雑草を通る経路は障害物が途中にあるが移動ロボットにとっては小さい障害であり，大きな負担にならない．雑草を通る経路の距離は短いので障害物を考慮しても，長い距離を通って迂回する経路より適切な経路だと考えられる．そのため，その環境の最短経路は雑草を通過する経路になる．距離と重みを考慮した最短経路問題をシミュレーションするためには，移動ロボットにとって大きな障害である池を大きい重みとして扱い，小さい障害である雑草を小さい重みとして扱う．コストを距離と重みの積としたとき，目標点へ向かうためのコストは池を通過する経路が最も大きくなり，雑草を通過する経路が最も小さくなる．これにより雑草を通過する経路が最短経路だとシミュレーション

上で計算することができる．本研究では距離と重みを考慮した最短経路を求めるため障害物に重みをつける．

本研究では，メモリ効率の良いアルゴリズムを開発するために Asano と Doerr[**1**] のアルゴリズムを導入している．Asano と Doerr[**1**] のアルゴリズムはまず格子グラフをいくつかの小格子グラフに分割する．ここで各小格子グラフのふちにある点を区別して境界点と呼ぶとする．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，各頂点への最短距離を計算する．その後，境界点への最短距離のみを記憶しておき，境界点以外の内部の点への最短距離は忘れる．分割された小格子グラフ全てに対してダイクストラ法を行う処理をスキャンと呼ぶことにする．このスキャンを高々境界点の総数分繰り返すことで始点から終点までの最短距離が求まる．

最短経路を出力するためには，まず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，終点から小格子グラフ内の頂点への最短距離を計算する．始点からの最短距離が記憶されている境界点のうち，終点を含む小格子グラフと隣接した小格子グラフに属している境界点を経由点と呼ぶことにする．終点から経由点までの経路を確定した後，経由点が属しているもう一つの小格子グラフに対して，ダイクストラ法を実行し，経由点から小格子グラフ内の頂点への最短距離を計算する．同様な処理を始点まで繰り返すことで始点から終点までの最短経路を求める．

本研究では，まず平面上に斜線を含む格子グラフを作成する．これにより，容易に平面上を通る経路を出力することができる．また最初に平面全体の格子グラフを全て作成するのではなく，導出過程で格子グラフを適宜作成する．そのため省メモリなアルゴリズムが期待できる．

格子グラフを作成後，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．そして，隣接する頂点間の辺の重みを計算された頂点の重みを用いて求める．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．その後，Asano と Doerr[**1**] のアルゴリズムを導入することで最短経路を求める．距離情報を小格子グラフの境界点だけ管理しているので，少ないメモリで始点から終点への最短経路を求めることができる．

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厳密な最短経路の近似である．視覚的に最短経路に見えるようにするために，向きを変える回数が多い経路を選択するようにする．

また，本研究で考案した手法が厳密解を求める手法に近いものか検証するために計算機で実験を行った．本手法は格子の本数を増やせば増やすほど，実際の最短経路に近い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アルゴリズムである．そこでまず実際の最短経路に近い経路を出力する格子の本数を調査する実験を行った．あらかじめ厳密な最短経路がわかる実験環境を設定し，厳密な最短経路に近い経路を出力する時の格子の本数を調査した．つぎに障害物の重みを変えることで，経路がどのように変化するか実験を行った．始点と終点を結ぶ直線上の経路に重みがついた障害物がある実験環境を設定して，出力した経路を調査した．

計算機実験により本手法はある程度実用的な最短経路を求めることができるアルゴリズ

ムであることが示された．本手法は$O(n^{\frac{2}{3}})$領域を用い$O(n^{1+\frac{2}{3}}\log n)$時間で動作する．しかし，本手法は複雑なアルゴリズムを用いずに最短経路を求めるものなので，得られる経路は格子上の経路であって実際の厳密な最短経路とは異なる．実際の厳密な最短経路を求めるためには多くのメモリが必要になる．精度が高い経路が計算できる複雑でないアルゴリズムを見つけることが今後の課題である．

目 次

第 1 章　はじめに **1**
1.1　研究目的 2
1.2　重みつき障害物 4
1.3　問題の定義 5
1.4　本論文の構成 5

第 2 章　格子グラフ上での最短経路アルゴリズム **6**
2.1　本研究のアルゴリズムの概要 6
2.2　小格子グラフに分割して解くアルゴリズム 7
2.2.1　最短距離の計算 7
2.2.2　最短経路の出力 10
2.3　格子グラフの作成 13
2.4　辺の重みの計算 14
2.5　ダイクストラ法による最短距離の計算 16
2.6　経路の改善 16
2.7　本手法の作業領域と計算時間 18

第 3 章　実験：格子グラフ上で最短経路を求める手法の検証 **19**
3.1　実験 1：格子の本数の調査 19
3.1.1　実験環境 20
3.1.2　結果 21
3.2　実験 2：重みによる経路の変化 25
3.2.1　実験環境 25
3.2.2　結果 26
3.3　実験のまとめ 28

第 4 章　おわりに **29**
4.1　まとめと今後の課題 29

謝辞 **30**

第1章　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．高濃度に汚染された原子炉建屋での作業は人間に代わって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 1.1** に示す．

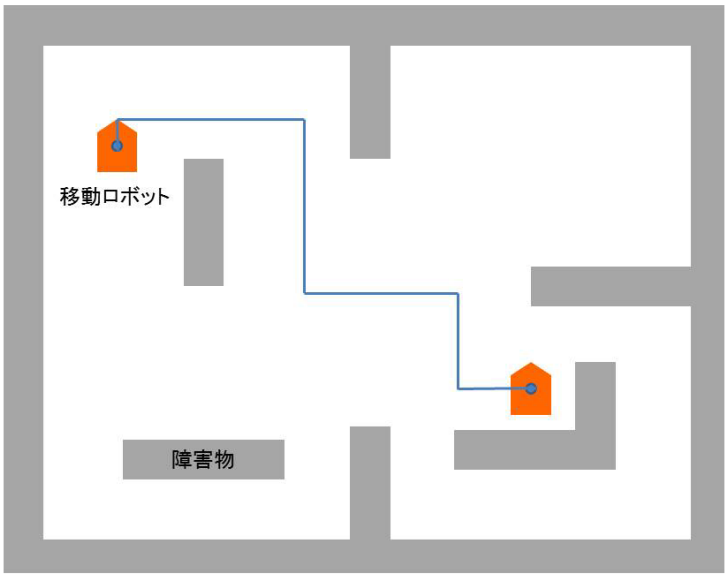

図 1.1: ロボットの経路計画

最短経路を見つけることは障害物を含む平面上の 2 点間の最短経路を求める問題として計算幾何学で扱われている．この問題は平面上に多角形の障害物がいくつか与えれているとき，指定された始点と終点を結ぶ最短経路を見つける問題である．Hershberger と Suri[**3**] は $O(N \log N)$ 時間で平面上の最短経路問題を計算するアルゴリズムを提案した．ただし，N は障害物の頂点数である．障害物と平面上の 2 点がある図を**図 1.2** に示す．

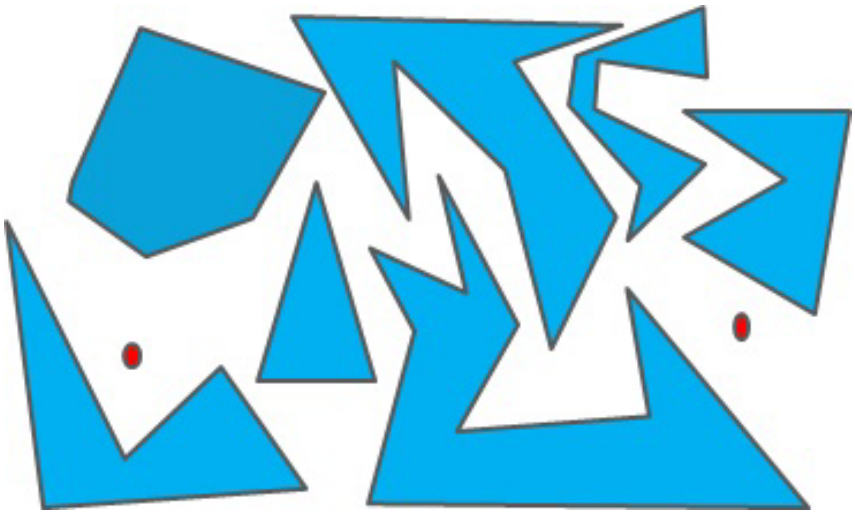

図 1.2: 障害物と平面上の 2 点

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[**1**] 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アルゴリズム [**4**] が提案されている．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，この手法は複雑で計算時間が長いのが欠点である．そこで本研究では，格子グラフを用いることで障害物を考慮した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

価を払うことで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．また，省メモリのためには格子グラフ全体を記憶するのではなく，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．

また本研究では，使用する計算機を一般的な RAM モデルを想定している．そのため 1 単位時間で 1 ワード，$\log n$ ビットのサイズを読み書きできる．

1.2　重みつき障害物

現場では距離と重みを考慮した経路を選択しなければならない状況がある．例えば，**図 1.3** のように移動ロボットが走行する目標地点までの最短経路を考える．地図上に障害物がなければ最短経路は目標地点へ直線に進む経路になる．ここでは，移動ロボットは池を通過することが困難であると仮定すると，目標地点への直線上に池があるので直線に進む経路は最短経路にならない．それ以外で予想する最短経路は，遠回りだが障害物がない経路と距離は短いが道の途中に雑草がある経路の 2 通りの経路であると仮定する．この場合の最短経路は雑草を通過する経路になる．雑草を通る経路は障害物が途中にあるが移動ロボットにとっては小さい障害であり，大きな負担にならない．雑草を通る経路の距離は短いので障害物を考慮しても，長い距離を通って迂回する経路より適切な経路だと考えられる．そのため，**図 1.3** の最短経路は雑草を通過する経路になる．

距離と重みを考慮した最短経路問題をシミュレーションするためには，移動ロボットにとって大きな障害である池を大きい重みとして扱い，小さい障害である雑草を小さい重みとして扱う．コストを距離と重みの積としたとき，目標点へ向かうためのコストは池を通過する経路が最も大きくなり，雑草を通過する経路が最も小さくなる．これにより雑草を通過する経路が最短経路だとシミュレーション上で計算することができる．本研究では距離と重みを考慮した最短経路を求めるため障害物に重みをつける．

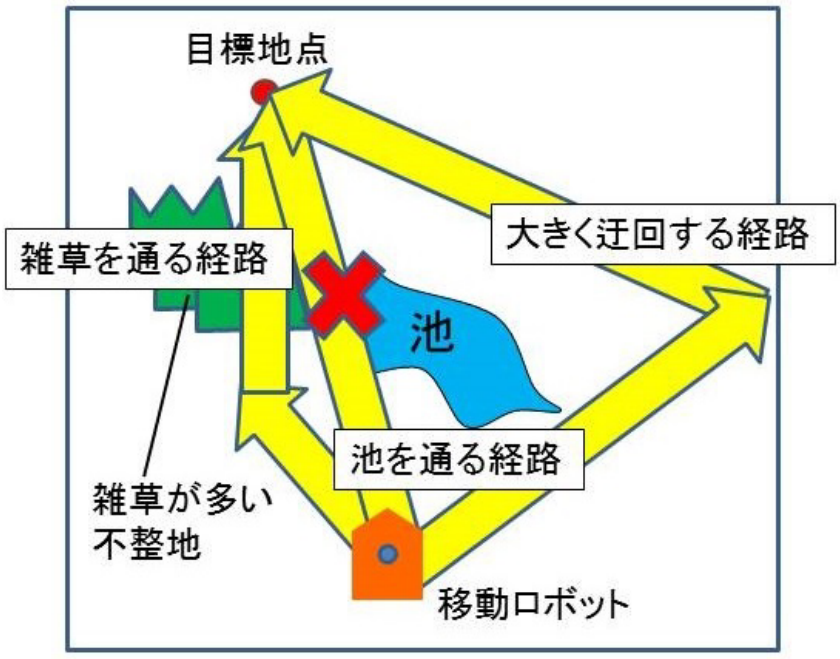

図 1.3: 重みを考慮した経路

1.3　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がついているものとする．平面上には始点 s, 終点 t の 2 点があり，s から t への最小コストの経路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．**図 1.4** に示したのは問題の一例である．

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．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，ロボットの幅を考慮した経路を出力することができる．

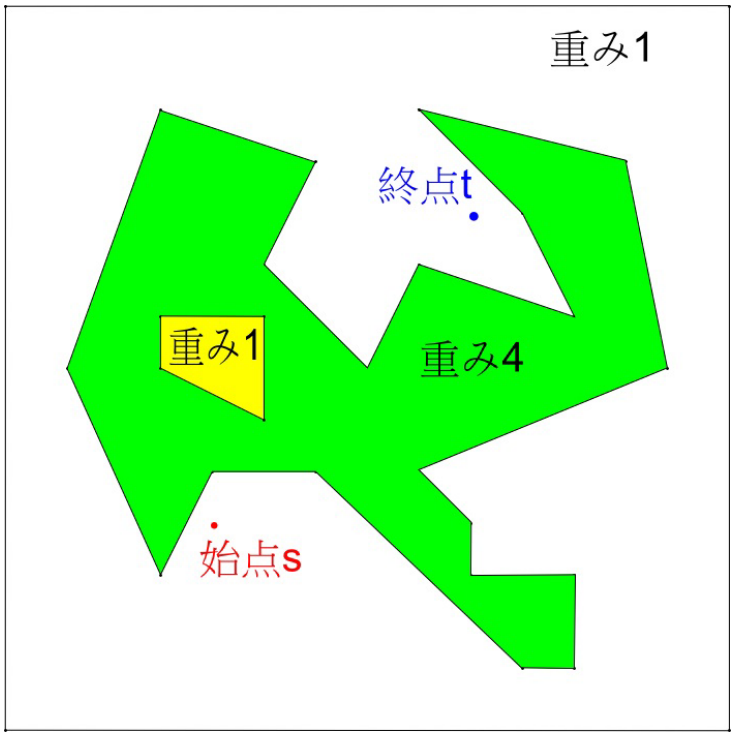

図 1.4: 問題の図

1.4　本論文の構成

本論文は次の構成からなる．第 2 章では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手法について述べる．また，Asano と Doerr[1] のアルゴリズムを導入したことについても述べる．第 3 章では前章の手法を実装し，計算機実験により検証した結果を述べる．最後に，第 4 章で本論文をまとめる．

第2章　格子グラフ上での最短経路アルゴリズム

本章では，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. 必要に応じて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経路を求める．

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　小格子グラフに分割して解くアルゴリズム

本研究では，メモリ効率の良いアルゴリズムを開発するために Asano と Doerr[1] のアルゴリズムを導入している．ここでは格子グラフを小格子グラフに分割して最短経路を求めるアルゴリズムについて説明する．

2.2.1　最短距離の計算

[1] 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．**図 2.1** 参照．S の添え字の 10 の位の数が行番号で 1 の位の数が列番号を表している．各小格子グラフのふちにある点を区別して境界点と呼ぶとする．境界点は各小格子グラフの境界に位置しているため，隣接した二つの小格子グラフに属している．(三つ，四つの小格子グラフに属している点もある．)

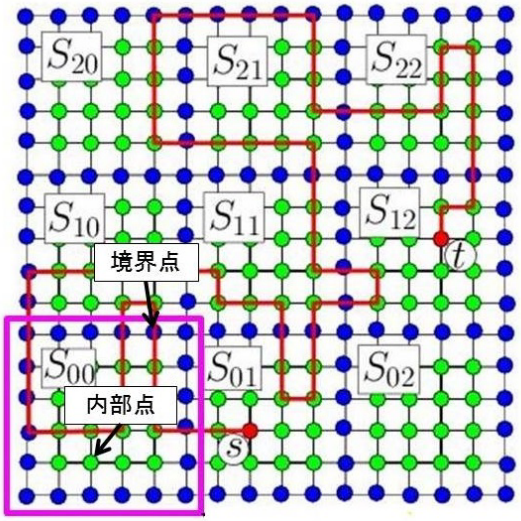

図 2.1: 最短距離の計算手順 1

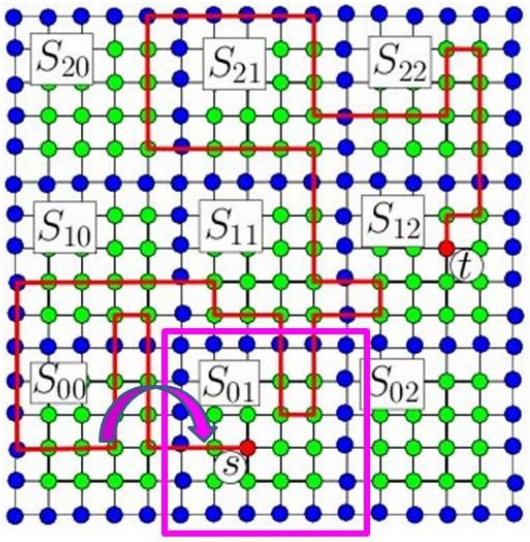

図 2.2: 最短距離の計算手順 2

図 2.1 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，各頂点への最短距離を計算する．その後，境界点への最短距離のみを記憶しておき，境界点以外の内部の点への最短距離は忘れる．その後，境界点が属しているもう一つの小格子グラフにおいてダイクストラ法で計算された境界点への最短距離を用いて，隣接した小格子グラフでダイクストラ法を実行する．**図 2.2** 参照．分割された小格子グラフ全てに対してダイクストラ法を行う処理をスキャンと呼ぶことにする．このスキャンを高々境界点の総数分繰り返すことで始点から終点までの最短距離が求まる．高々境界点の総数分繰り返すのは，**図 2.3** 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である．戻る回数は高々境界点の総数分になる．一連の手順を **Algorithm1** に示す．

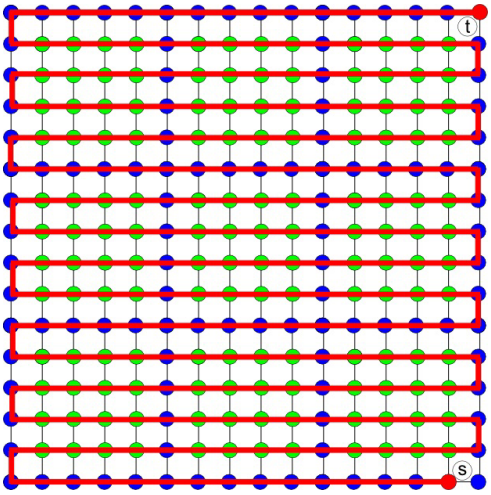

図 2.3: 一度ダイクストラ法を実行した小格子グラフに戻ってくる経路になる場合

Algorithm 1 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: 辺の重みをもち，始点 s と終点 t を含む $\sqrt{n} \times \sqrt{n}$ サイズの格子グラフ G（n：頂点数，$\sqrt{n}$：整数），一辺の分割数 k

Output: 始点 s から終点 t までの最短距離

//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**
    //境界点の距離情報だけ配列 T へ移す
    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

各小格子グラフの頂点は $\frac{\sqrt{n}}{k} \times \frac{\sqrt{n}}{k} = \frac{n}{k^2}$ 個ある．ダイクストラ法の計算時間は $O(n^2)$ なので各小格子グラフにおいての計算時間は $O(\frac{n^2}{k^4})$ である．小格子グラフは全部で k^2 個あるので，1 回のスキャンの計算時間は $O(\frac{n^2}{k^4}) \times k^2 = O(\frac{n^2}{k^2})$ 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計算時間は $O(\frac{n^2}{k^2}) \times k\sqrt{n} = O(\frac{n^{2+\frac{1}{2}}}{k})$ である．

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グラフに移ったあと始点から境界点までの距離情報が作業領域として必要となるので，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．

2.2.2 最短経路の出力

最短経路を出力するためには，まず**図 2.4** 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，終点から小格子グラフ内の頂点へ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**式 (2.1)** で確かめる．

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．

$$D(s, v_i) + d(v_i, t) = d(s, t) \tag{2.1}$$

始点からの最短距離が記憶されている境界点のうち，終点を含む小格子グラフと隣接した小格子グラフに属している境界点を経由点と呼ぶことにする．**式 (2.1)** を満たす境界点が経由点の候補である．境界点が複数あった場合，その中で $d(v_i, t)$ が一番大きい値をもつ境界点が経由点である．その境界点を選ぶことで，途中境界点を通る経路であっても，隣の小格子グラフに到達するまで経路探索を続けることができる．これは本研究で改良した部分である．終点から経由点までの経路を確定した後，**図 2.5** のように経由点が属しているもう一つの小格子グラフに対して，ダイクストラ法を実行し，経由点から小格子グラフ内の頂点への最短距離を計算する．同様な処理を始点まで繰り返すことで始点から終点までの最短経路を求める．

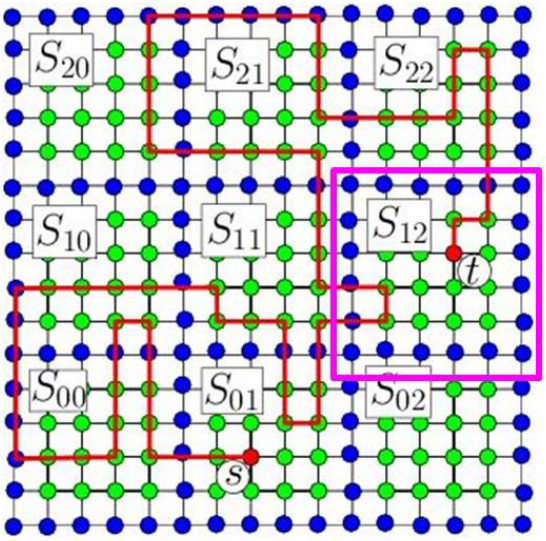

図 2.4: 最短経路の出力手順 1

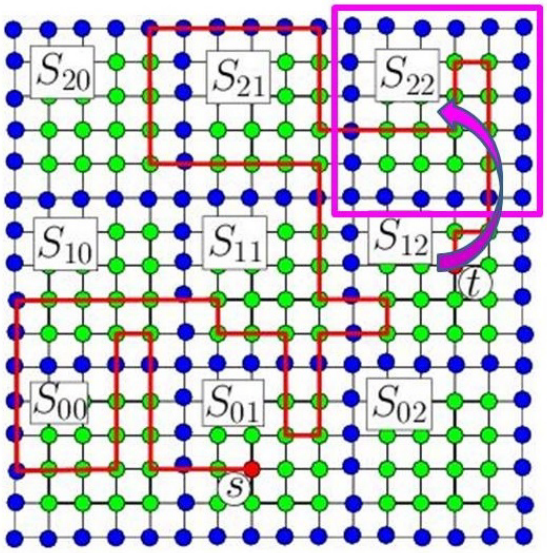

図 2.5: 最短経路の出力手順 2

最短経路を出力する手順を **Algorithm2** に示す.

計算時間は，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.

Algorithm 2 格子グラフ上の 2 点間の最短経路を出力するアルゴリズム

Input: 辺の重みをもち，始点 s と終点 t を含む $\sqrt{n} \times \sqrt{n}$ サイズの格子グラフ G（n：頂点数，$\sqrt{n}$：整数），一辺の分割数 k，始点 s から終点 t までの最短距離，始点 s から各境界点までの最短距離 C[i][j]

Output: 始点 s から終点 t までの最短経路

//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までの距離を配列 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:    //終点から小格子グラフ内の頂点への最短距離を計算する
7:    while S_ij の内部の非選択の頂点がある do
8:       未確定の頂点から T[i][j] が最も小さい頂点を選ぶ
9:       選択した点に印をつける
10:      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
11:         T[q] = T[p] + w
12:      end if
13:   end while
14:   for S_ij の中の境界点 do
15:      if C[i][j] + T[i][j] == leng then
16:         T[i][j] が最大な頂点を選ぶ
17:      end if
18:   end for
19:   //Backtracking
20:   if leng 値が格納されている点の周囲の点 T[p] + w == leng then
21:      w の向きに経路を出力する
22:      leng = leng − w
23:   end if
24:   if 隣の小格子グラフに移る直前の経由地の境界点 then
25:      境界点が属するもう片方の小格子グラフの番号を求める
26:      上の指示（ダイクストラアルゴリズム）に戻る
27:   end if
28: end while
```

2.3　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一定の本数分作成する．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，頂点と頂点を結ぶ直線分を辺とする．このようにしてできたグラフを本稿では格子グラフと呼ぶ．これにより，容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 2.7** に示す．また最初に平面全体の格子グラフを作成するのではなく，必要に応じて部分的に小格子グラフを作成する．そのため省メモリなアルゴリズムが期待できる．作成した小格子グラフを**図 2.6** に示す．

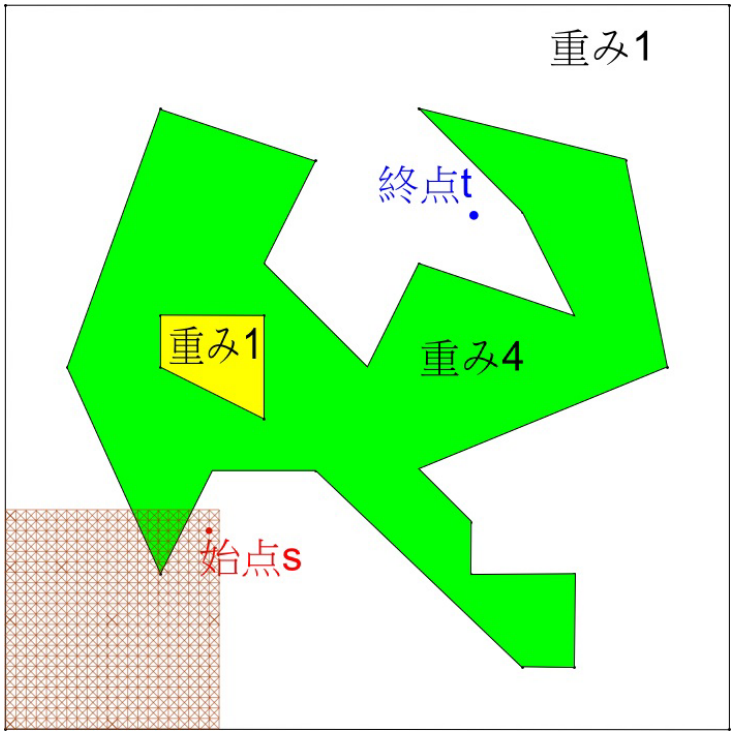

図 2.6: 小格子グラフ作成

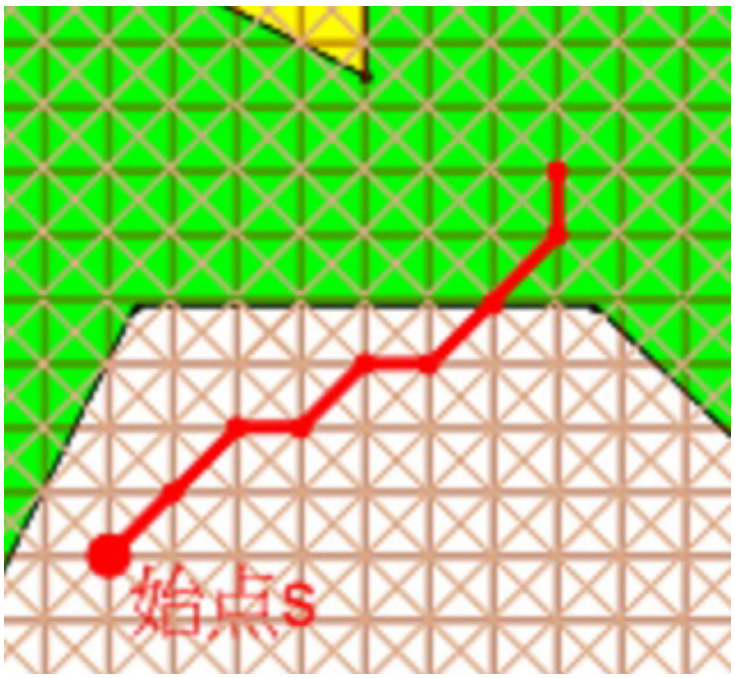

図 2.7: 経路を辿る様子

2.4 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）を頂点の重みを用いて求める．これにより障害物の重みを考慮した経路を実現している．計算式は**式 (2.2)**，**式 (2.3)** に示す．

- 2 点の頂点が上下左右に位置している場合

$$\text{2 点間の辺の重み（コスト）} = \frac{\text{自分の頂点の領域の重み} + \text{相手の頂点の領域の重み}}{2} \tag{2.2}$$

- 2 点の頂点が斜めに位置している場合

$$\text{2 点間の辺の重み（コスト）} = \frac{\text{自分の頂点の領域の重み} + \text{相手の頂点の領域の重み}}{2} \times \sqrt{2} \tag{2.3}$$

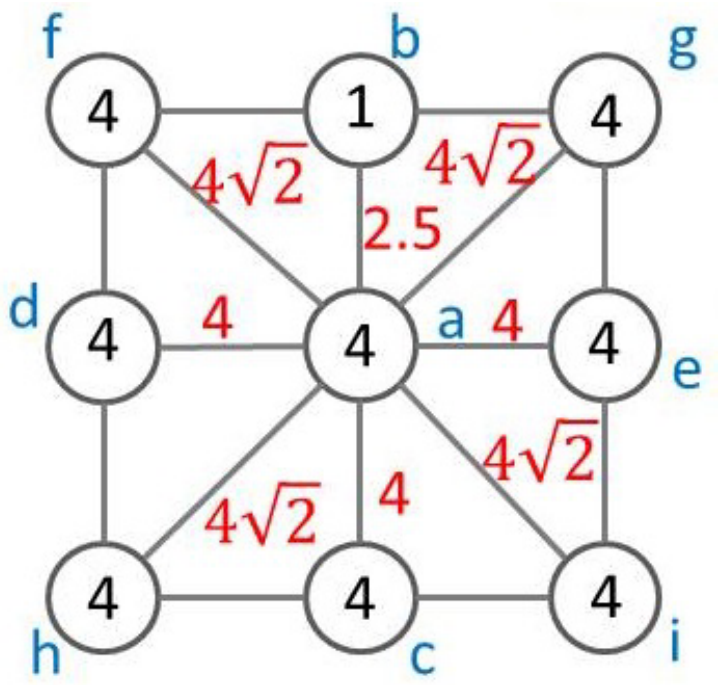

図 2.8: 頂点の重みと辺の重み

例えば a と b を結ぶ辺の重み（コスト）は

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

となり，a と f を結ぶ辺の重み（コスト）は

$$\frac{4+4}{2} \times \sqrt{2} = 4\sqrt{2}$$

となる．

しかし，**図 2.9** 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．

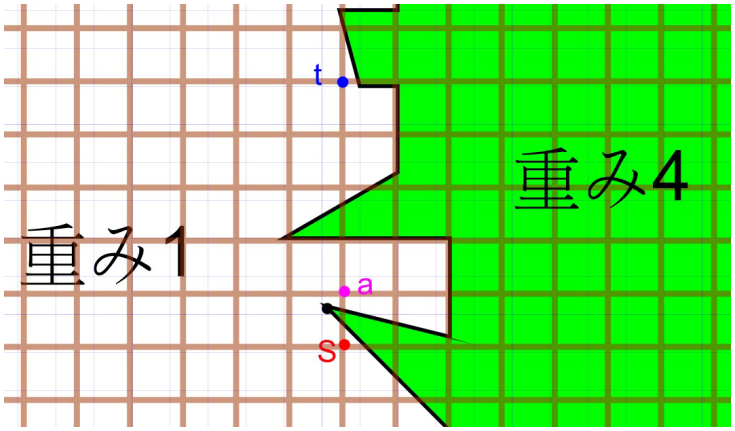

図 2.9: 辺の重みの近似

2.5 ダイクストラ法による最短距離の計算

辺の重みが求まった後は，Asano と Doerr[**1**] のアルゴリズムを導入することで最短経路を求める．頂点の重みの計算，辺の重みの計算，最短距離の計算は格子グラフを作成するたびに行う．そのため距離情報を小格子グラフの境界点だけで管理しているので，少ないメモリで始点から終点への最短経路を求めることができる．

2.6 経路の改善

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厳密な最短経路の近似であり，マンハッタン距離（L1 距離）に近い経路である．例えば**図 2.10** 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，厳密な最短経路とは違う．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．

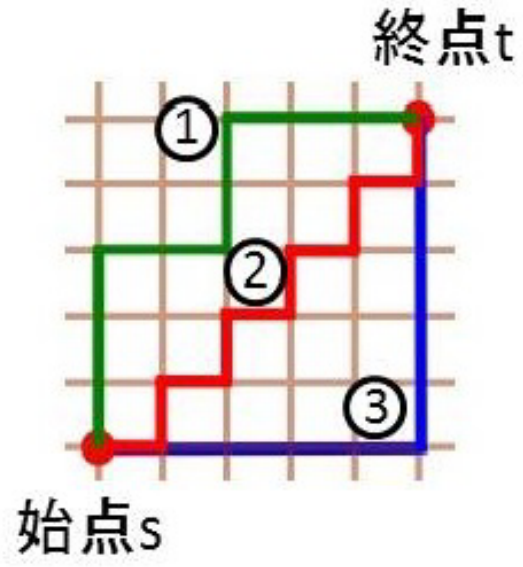

図 2.10: マンハッタン距離

視覚的に最短経路に見えるようにするために，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 2.10** の場合，s から t への最短経路候補は ①，②，③ とある．本アルゴリズムでは ② のコースのような経路を選択する．その結果，格子の本数を増やした時，② の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経路で結んでいるようにみえる．そのアルゴリズムを **Algorithm3** に示す．**Algorithm3** 導入前の経路を**図 2.11** に，導入後の経路を**図 2.12** に示す．

Algorithm 3 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

Input: 辺の重みをもち，始点 s と終点 t を含む $\sqrt{n} \times \sqrt{n}$ サイズの格子グラフ G（n：頂点数，$\sqrt{n}$：整数），始点 s から各頂点までの最短距離
Output: 始点 s から終点 t までの最短経路の中で向きを変える回数が多い経路

```
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
```

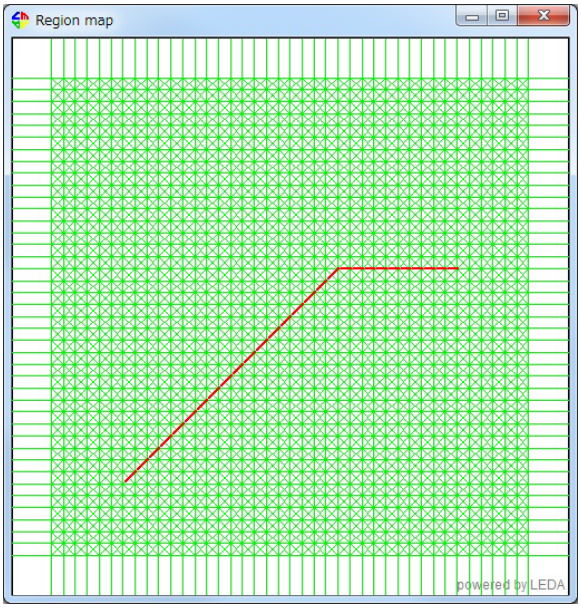

図 2.11: **Algorithm3** 導入前

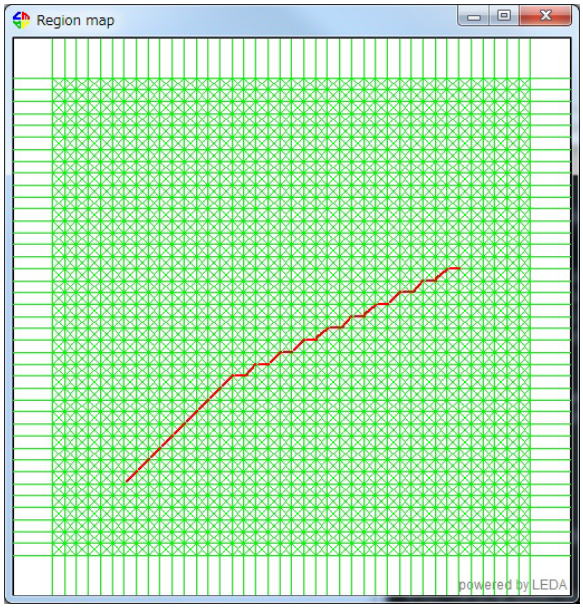

図 2.12: **Algorithm3** 導入後

2.7 本手法の作業領域と計算時間

辺の数を m，頂点の数を n としたときダイクストラ法の計算時間は，$O(m + n \log n)$ 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）$m \approx 4n$ となるので，それぞれダイクストラ法の計算時間に代入すると計算時間は $O(n \log n)$ となる．さらに小格子グラフ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 $O(n^{1+\frac{2}{3}} \log n)$ となる．作業領域はダイクストラ法の作業領域である $O(n)$ に，小格子グラフ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．よって，計算時間は長くなっているが，作業領域は小さくなり，メモリ効率は良くなっている．

第3章　実験：格子グラフ上で最短経路を求める手法の検証

考案したアルゴリズムが厳密解を求めるアルゴリズムに近い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，ドイツの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 [**2**].

3.1　実験1：格子の本数の調査

格子の本数を増やせば増やすほど，実際の最短経路に近い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．そこでまず実際の最短経路に近い経路を出力する格子の本数を調査する．

3.1.1 実験環境

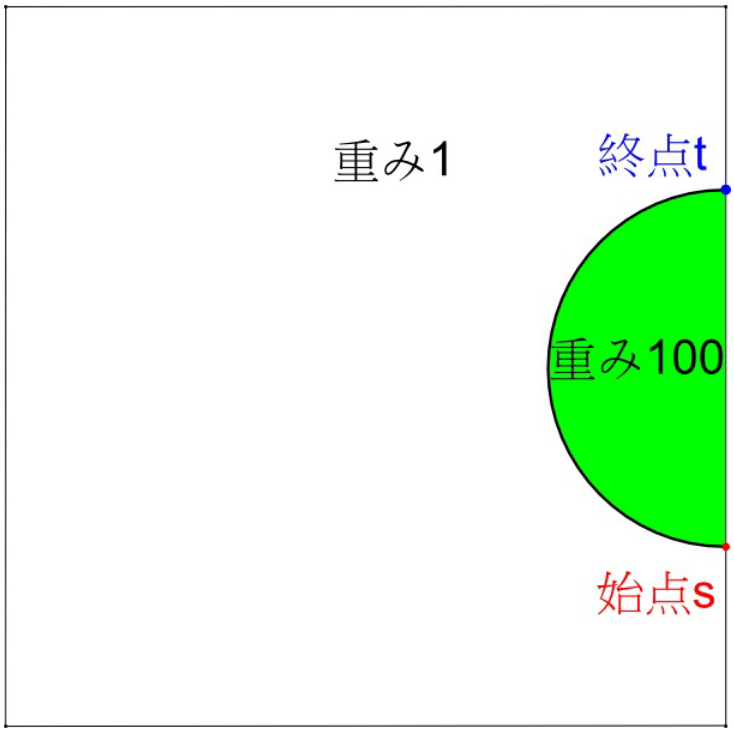

図 3.1: 実験 1 の実験環境

あらかじめ厳密な最短経路がわかる実験環境を設定した．**図 3.1** 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用意した．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 3.1** の上にひく格子の本数を変化させる．最短経路が円周を辿る経路になるので，その経路に近い経路を出力する時の格子の本数を調査する．

3.1.2 結果

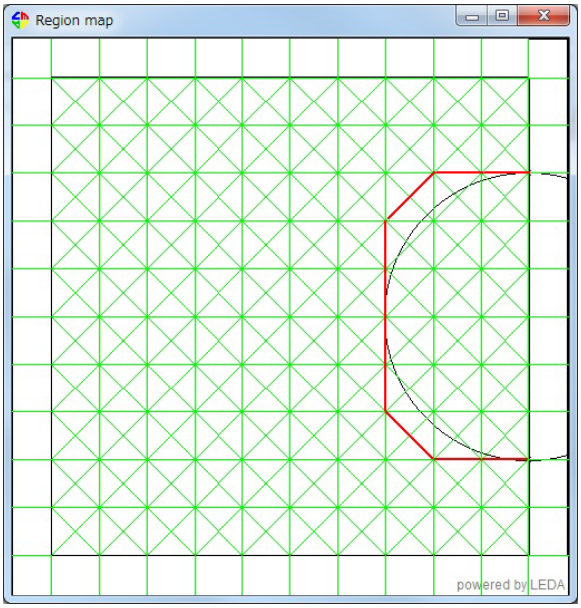

図 3.2: 格子 10 本

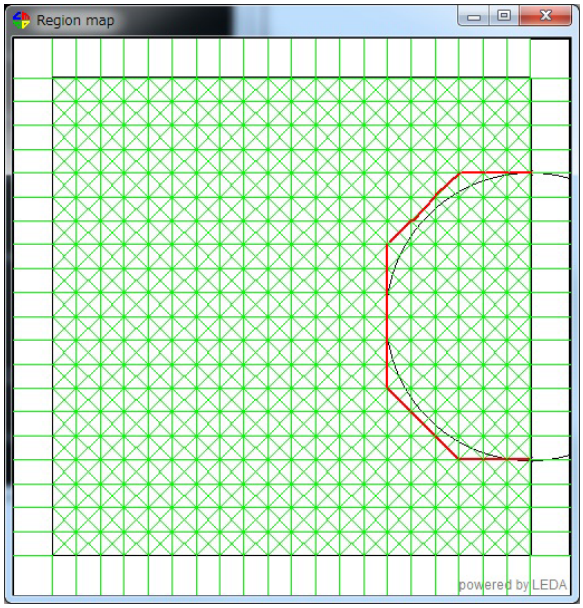

図 3.3: 格子 20 本

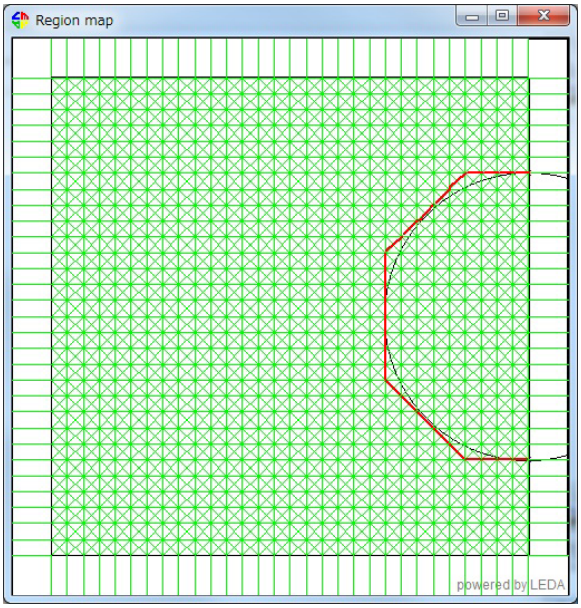

図 3.4: 格子 30 本

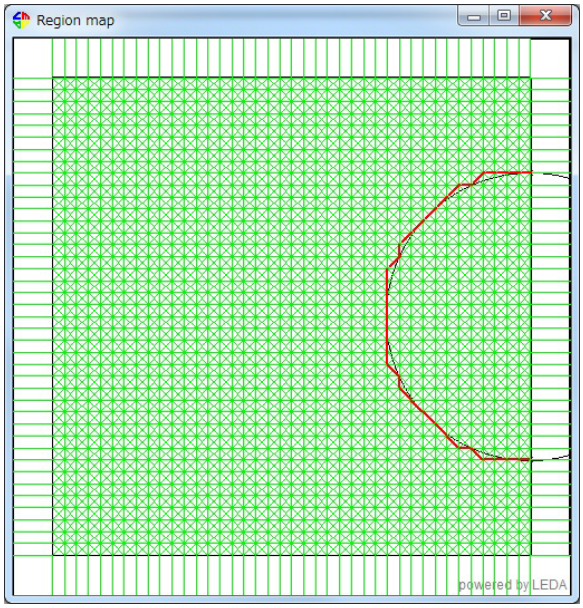

図 3.5: 格子 40 本

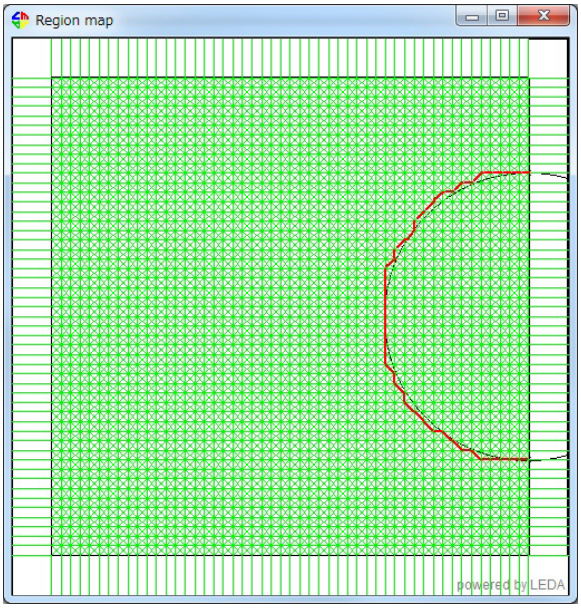

図 3.6: 格子 50 本

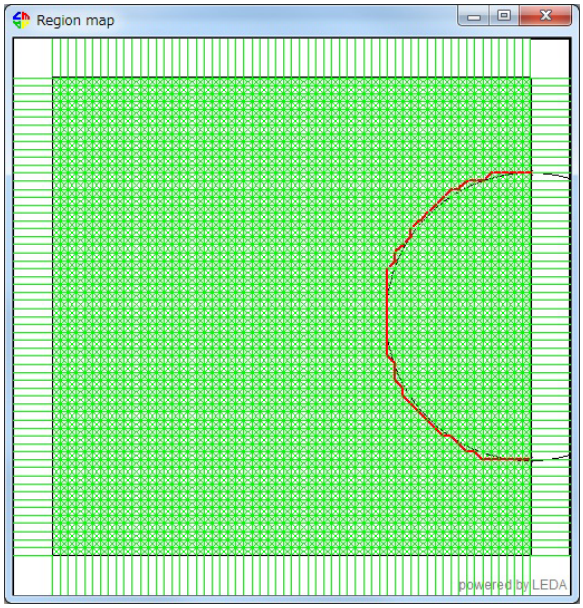

図 3.7: 格子 60 本

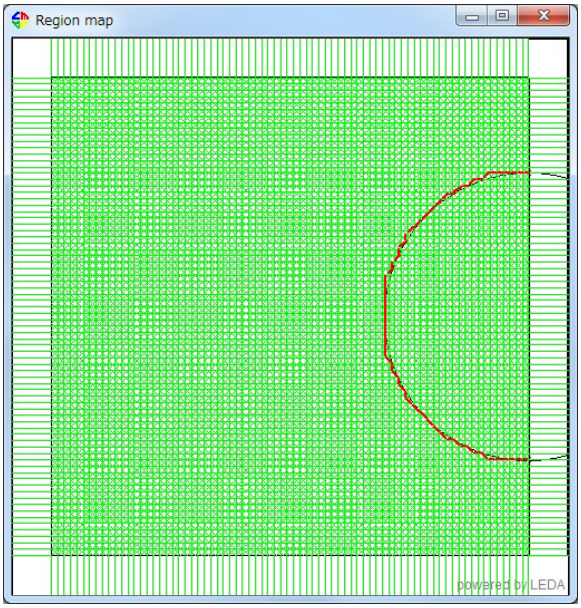

図 3.8: 格子 70 本

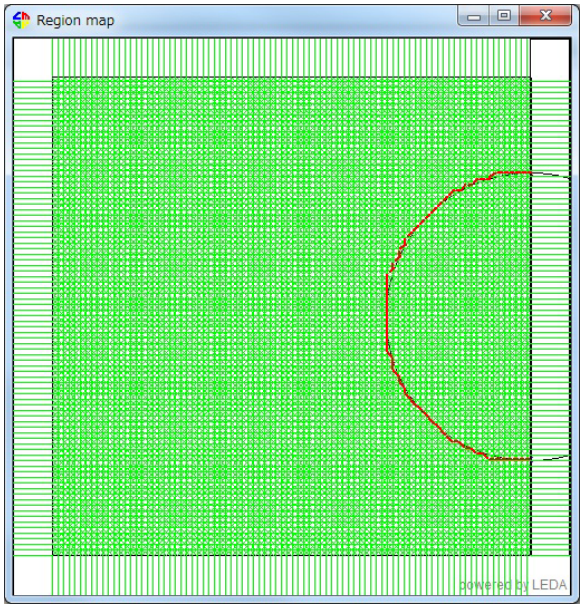

図 3.9: 格子 80 本

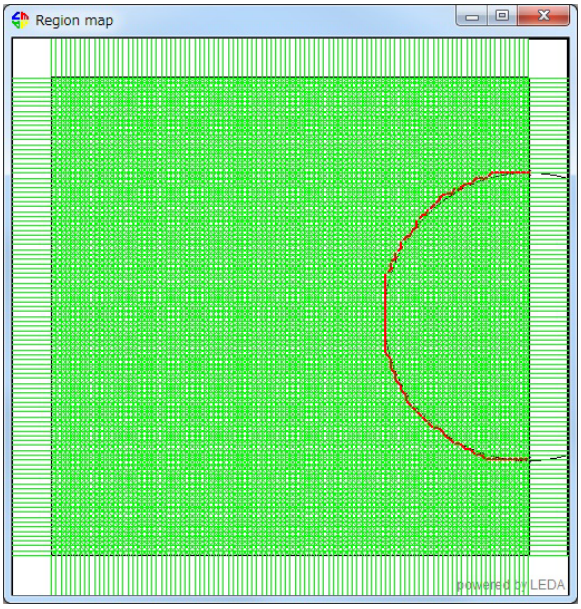

図 3.10: 格子 90 本

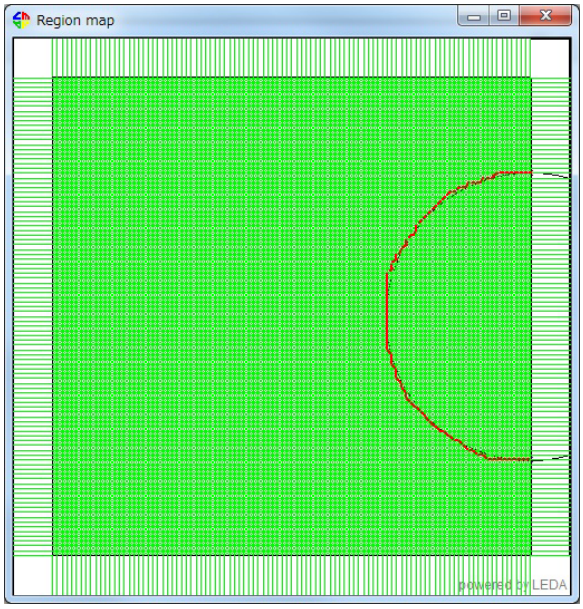

図 3.11: 格子 100 本

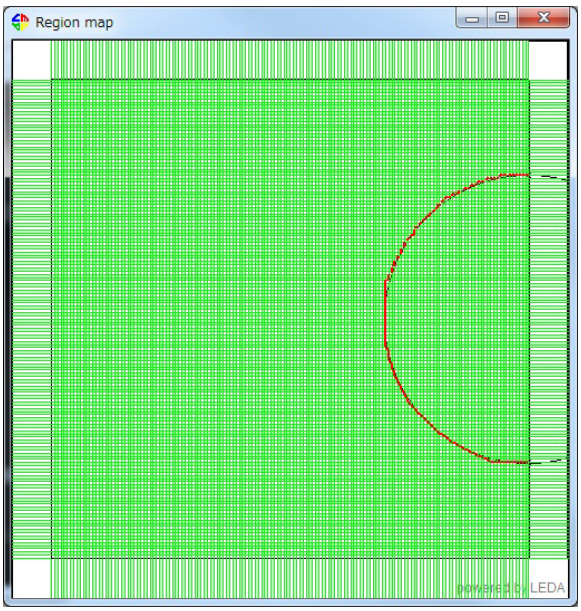

図 3.12: 格子 150 本

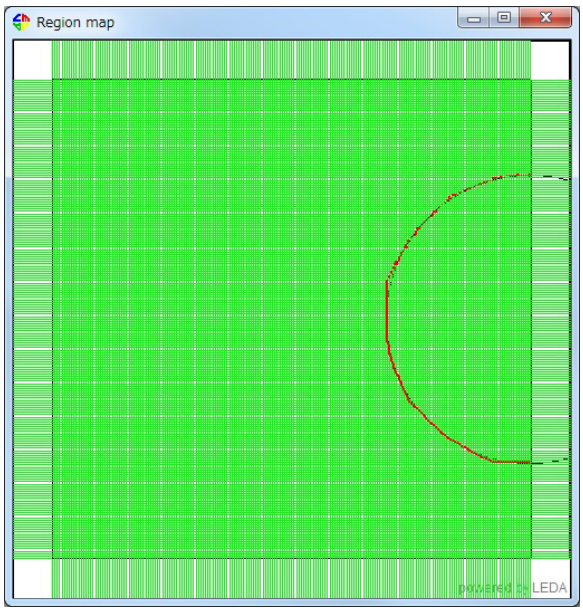

図 3.13: 格子 200 本

格子 10 本から 200 本までの経路を**図 3.2**〜**図 3.13** に示す．格子 10 本（頂点数：100）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おり，厳密な最短経路に近い経路にみえる．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．そしてそこから厳密な解の値（半円の半径 (180.0)）を引き，厳密な解との差を求めた．グラフにしたものを**図 3.14** に示す．

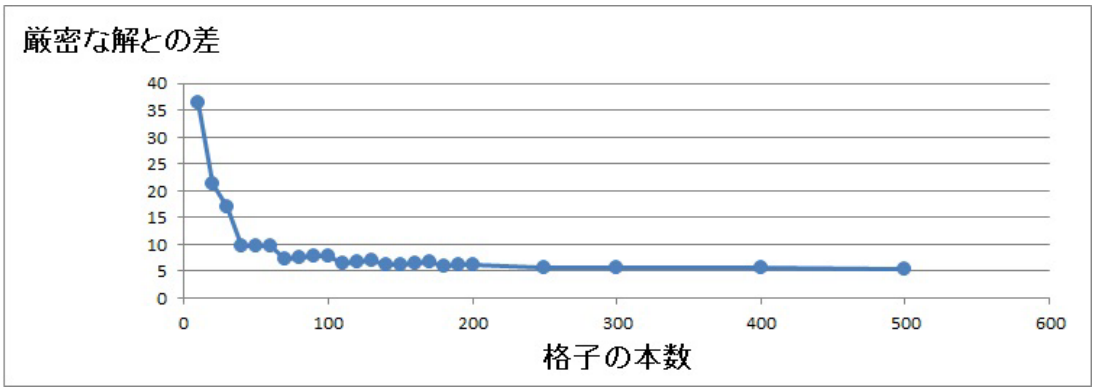

図 3.14: 格子の本数と厳密な解との差

格子の本数を増やしていくと差が縮まっている．よって格子の本数を増やせば，厳密な解との差が 0 に近い値になるので，厳密な最短経路に近づく．また本数を 500 本（頂点数：250000）と増やしても差が 0 にはならなかった．これは格子の本数を増やしても，厳密な解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，格子の本数を増やせばより円周を沿う経路になり厳密な最短経路に近い経路になるが，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解が出力されているため，以降の実験では格子の本数を 100 本とする．

3.2　実験 2：重みによる経路の変化

ある領域の重みを変化させることで最短経路がどのように変化するかを調べる実験を行う．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

始点と終点を結ぶ直線上の経路に重みがついた障害物がある実験環境を設定する．**図 3.15** に示す通りに，始点 s と終点 t の 2 点，重みの異なる領域を三つ用意した．中央よりやや左に位置する小さい四角形を多角形 P，中央の大きい多角形を多角形 Q とする．多角形 Q の内部で多角形 P 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．多角形 P の内部の領域の重みを 1〜100 まで変化させる．つぎに多角形 P の内部の領域の重みを 1，多角形 Q の外側の領域の重みを 1 としたときに，多角形 Q の内部で多角形 P には含まれない領域の重みを 1〜100 まで変化させる．

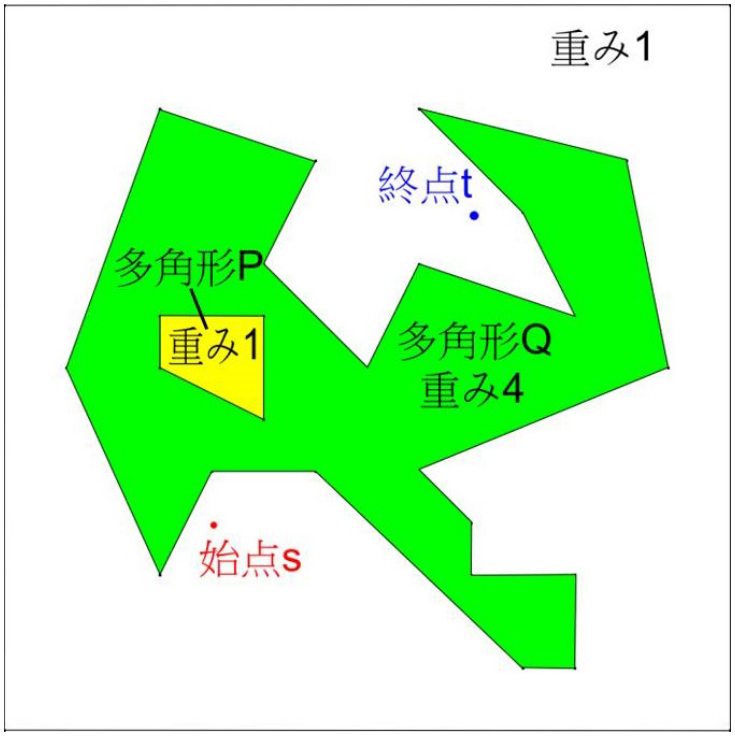

図 3.15: 実験 2 の実験環境

3.2.2　結果

多角形 P の内部の領域の重みを 1〜100 まで変化させたときは経路が 2 種類に分かれた．重みが 1 のときの経路を**図 3.16** に，重みが 2 より大きいときの経路を**図 3.17** に示す．始点と終点を直線で結ぶ経路のそばに重みが小さい領域がある場合はその領域を通る経路になっている．しかし，重みの値が大きくなると始点と終点を直線で結ぶ経路に近くなっている．

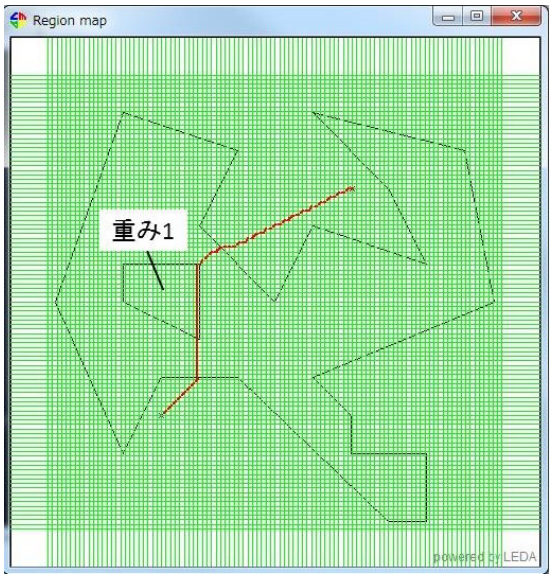

図 3.16: 重み 1 のとき

図 3.17: 重みが 2 より大きいとき

多角形 Q の内部で多角形 P には含まれない領域の重みを 1〜100 まで変化させたときは経路が 3 種類に分かれた．ひとつは**図 3.16** のときと同じ経路になる．重みが 1 のときの経路を**図 3.18** に，重みが 7 より大きいときの経路を**図 3.19** に示す．

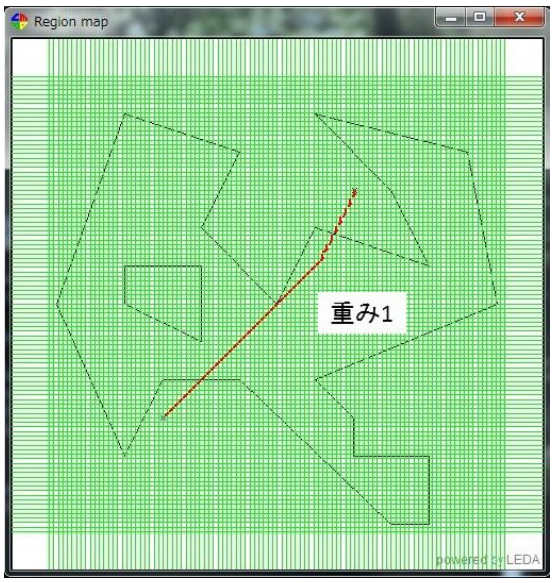

図 3.18: 重み 1 のとき

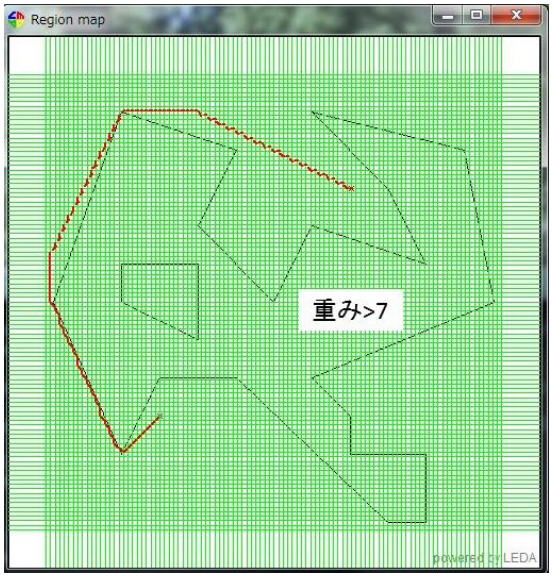

図 3.19: 重みが 7 より大きいとき

重みが 1 のときはすべての領域の重みが同じ値なので，始点と終点を直線で結ぶ経路になるはずだが，それに近い経路になっている．これは距離を近似している影響だと思われる．重みが 7 より大きいときは，迂回する経路になっている．そのため，距離が長い．

3.3　実験のまとめ

小さい重みをもつ領域があるときはその領域を通過する経路を，かなり大きい重みをもつ領域があるときはその領域を避けて迂回する経路を出力していた．そのため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していることがわかった．しかし，実験1において格子の本数を増やしても完全に厳密な最短経路にはならなかった．さらに実験2の重みが2より大きいときの経路が，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いたり，重み1の領域の方向へ離れたりを何度も繰り返す経路となっている．そのため，実際の経路とは少し異なっている．

第4章　おわりに

4.1　まとめと今後の課題

今回は少しずつ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．計算機実験により本手法はある程度実用的な最短経路を求めることができるアルゴリズムであることが示された．本手法は，$O(n^{\frac{2}{3}})$ 領域を用い $O(n^{1+\frac{2}{3}}\log n)$ 時間で動作する．また，本手法は複雑なアルゴリズムを用いずに最短経路を求めるものなので，得られる経路は格子上の経路であって実際の厳密な経路とは異なる．実際の厳密な最短経路を求めるためには多くのメモリが必要になる．精度が高い経路が計算できる複雑でないアルゴリズムを見つけることが今後の課題である．

謝辞

本研究を行うにあたり，全過程を通して丁寧かつ熱心な御指導を賜りました，北陸先端科学技術大学院大学 浅野哲夫学長，および情報科学研究科 上原隆平教授，大舘陽太助教に心より感謝致します．特に修論発表・論文・日常の議論を通じて多くの知識や示唆を頂きました．

また，本大学情報科学研究科の諸教官方のご指導に感謝するとともに厚く御礼申し上げます．

最後に，研究活動において常に温かい御助言を頂いた情報科学研究科 上原研究室の諸氏に深く感謝致します．

平成 27 年 2 月

1310057　早川裕真

参考文献

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

[2] 浅野哲夫，小保方幸次：”LEDA で始める C/C++プログラミング”，初版，サイエンス社，2002.

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

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