2018 年度　卒　業　論　文

動的な変化に対応する
経路探索の手法についての研究

指導教員：渡辺 大地 准教授

メディア学部 ゲームサイエンス プロジェクト
学籍番号　M0115236
津川 巧

2018 年 9 月

2018 年度　　卒　業　論　文　概　要

論文題目

動的な変化に対応する
経路探索の手法についての研究

メディア学部 学籍番号：**M0115236**	氏 名	津川 巧	指導 教員	渡辺 大地 准教授

キーワード	人工知能、経路探索、効率化、 アルゴリズム、リスト構造

経路探索はゲーム AI やカーナビゲーションシステムなどに用いられる。ゲーム AI における経路探索には、近年のゲームに多く実装されているもので、ナビゲーションメッシュという手法が適用されている。この手法は、起伏のある立体的な地形や敵対オブジェクトなどの位置関係を参照し、経路を計算する手法である。参照するデータは、ゲームを制作する際に用意しておく必要がある。また、地形のデータを作成するためには、先ほど述べたナビゲーションメッシュ等の自動化されたシステムを地形モデルに対し、用いることでスキャニングし、データを生成するほか、ゲームの制作者が決め打ちで作成する方法がある。以上のように経路探索はあらかじめ決まっているフィールドの情報を取得し、探索するといった経路に変化のないフィールドを探索することが一般的である。そのため、経路が変化するような探索には、変化のたびに再探索をする必要がある。しかし、毎回探索をしていては処理時間が増加し非効率的である。よって、経路の変化ごとにマップの再探索をするのではなく、効率的に探索処理をするというのが本研究の目的である。

手法としては、最短経路の経路情報をあらかじめ格納しておき、使い物にならないと判断した際に格納したデータから経路情報を削除する。そして、残った格納データから、一つの経路を参照することで最短経路を瞬時に叩き出すというものである。本手法を評価した結果、最短経路長と経路数がある程度少ないマップにおいて有効であることが判明した。また、D*アルゴリズムよりも分散値が低くバラつきが少ないことが分かった。

目 次

図 目 次

第 1 章

はじめに

1.1　背景と目的

経路探索はゲーム AI[1] や、カーナビゲーションシステム [2]、ロボット AI、交通シミュレーションや災害時の経路シミュレーションなどに用いられる。

ゲーム AI の分野では、高橋ら [3] はローグライクゲームにおける機械学習の研究を行った。また、藤井ら [4] は経路探索などによる人間らしい NPC を自律的に構成する手法を研究した。

近年のゲームに多く実装されているもので、ナビゲーションメッシュ [5] や NavMesh[6][7] という手法が適用されている。この手法は、起伏のある立体的な地形や敵対オブジェクトなどの位置関係を参照し、経路を計算する手法である。参照するデータは、ゲームを制作する際に用意しておく必要がある。参照データの生成では、ナビゲーションメッシュ等を用いた地形データのスキャニングによる自動生成の他、ゲーム制作者が手動で決め打ち生成する方法がある。

カーナビゲーションシステムの分野では、櫻場ら [8] は粘菌アルゴリズムを拡張した力学系による経路探索を提案した。

ロボット AI の分野では、鈴木ら [9] は車両型ロボットが滑らかに走行できるような手法を研究した。また、高松らは [10] は 3 次元迷路空間を自律走行するロボットが、視線スキャンによって

曲がり角を検出するアルゴリズムを研究した。

交通シミュレーションの分野では、福田ら [11] は階層化された道路ネットワークを用いた経路探索手法を提案した。また、丸ら [12] は車両のレーン変更を考慮した経路探索方式を開発した。

災害時の経路シミュレーションの分野では、古次ら [13] は粘菌アルゴリズムとダイクストラ法を比較し、粘菌アルゴリズムから得られる情報が有効であることを示した。また、梅木ら [14] は災害時の混雑情報を考慮した避難所決定手法の提案をした。

ここで、カーナビゲーションシステムやロボット AI 等に用いられる手法として、ダイクストラ法 [15][16] をはじめとしたアルゴリズムが用いられている。このダイクストラ法というアルゴリズムは、ノード間におけるコストから最適な経路を計算するものである。乗換案内 [17][18] もこのダイクストラ法を使用しており、駅間における交通費や、所要時間をコストに対応付けることで経路を探索している。ダイクストラ法の一般的な模式図を表したものが、図 1.1 である。円と円を結ぶ直線に付随して示している数値が、コストに当たる。

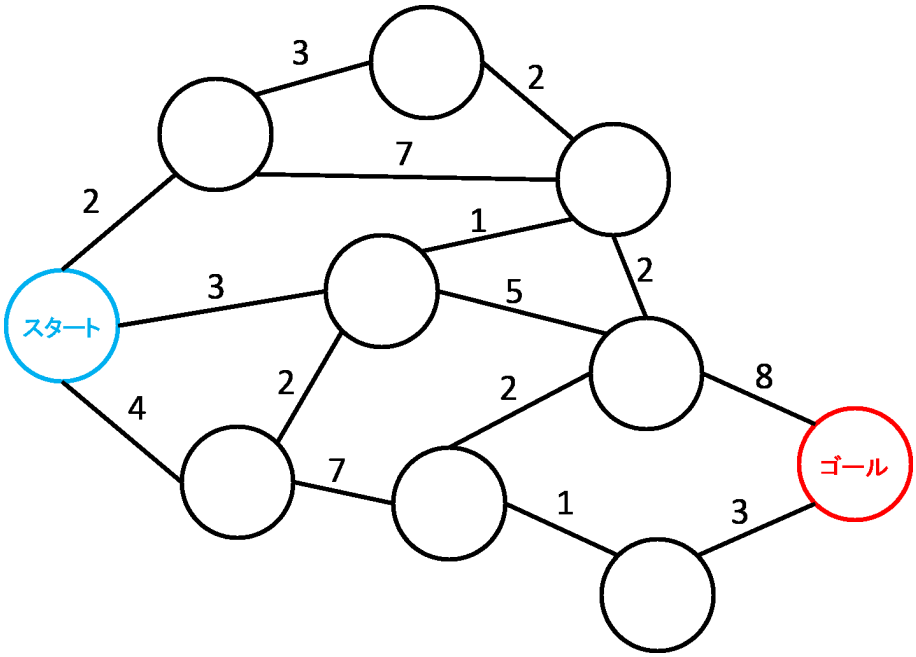

図 1.1　ダイクストラ法模式図

以上のように、経路探索はあらかじめ決まっているフィールドの情報を取得し探索するといっ

た、経路に変化のないフィールドを探索することが一般的である。そのため、経路が変化するような探索には、変化のたびに再探索をする必要がある。しかし、毎回探索をしていては処理時間が増加し非効率的である。よって、経路変化時の効率的な探索処理の手法を研究することを本研究の目的とする。

動的な変化に対応する経路探索の手法は、Game Programming Gems 第 5 巻にて、動的 A*(D*)[19] というアルゴリズムが紹介されている。しかし、この手法では特定の場面において冗長な処理を行っていることがある。したがって、本研究ではより効率的な処理を行う事を目的とする。

本研究の手法としては、最短経路の経路情報をあらかじめ格納しておき、使い物にならないと判断した際に格納したデータから経路情報を削除する。そして、残った格納データから、一つの経路を参照することで最短経路を瞬時に叩き出すというものである。本手法を評価した結果、最短経路長と経路数がある程度少ないマップにおいて有効であることが判明した。また、D*アルゴリズムよりも分散値が低くバラつきが少ないことが分かった。

1.2 論文構成

本論文は全 5 章にて構成する。構成は 2 章にて探索アルゴリズムについて述べ、3 章では提案手法について述べる。また、4 章にて評価と分析について述べ、そして 5 章にてまとめを述べる。

第 2 章

経路探索の先行研究

本章では、経路探索の成功手法について説明する。2.1 節では、探索アルゴリズムについて説明し、2.2 節では、現状の問題点について説明する。

本研究では経路として通過できるものは空白マス、通過が出来ないマスは壁マスと定義する。また、以下で取り上げるマップは複数のマスによって成り立つタイリングマップであり、マスはそれぞれスタートマス、ゴールマス、空白マス、壁マスの 4 種類の属性がある。進める方向は、上下左右のマスのいずれかで、斜め方向や 2 マス以上の移動は考慮しない。

2.1　探索アルゴリズム

経路探索とは、スタート地点からゴール地点までの経路をコンピュータでの計算によって求めるものである。この経路というものは、一般的には最適な経路であることが多く、特定のポイント間におけるコストに基づいて計算がなされている。

2.1.1　ダイクストラ法

以下に示す図 2.1～2.3 は経路探索アルゴリズムの一つであるダイクストラ法を迷路型のマップに表したもの [20] である。

はじめに、壁とルートのみを設けたマップにスタートのマスとゴールのマスを設置する。その様子を表しているのが、図 2.1 である。

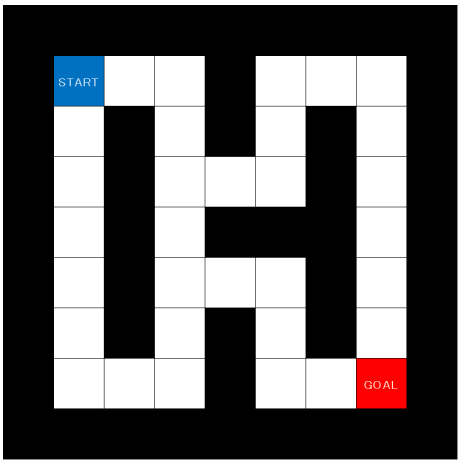

図 2.1　探索対象マップ

このマップのスタート地点からすべての通過できる上下左右のマスについて、数字を振っていく。このとき、すでに数字の振られたマスには数字を上書きすることはない。その様子が図 2.2 である。

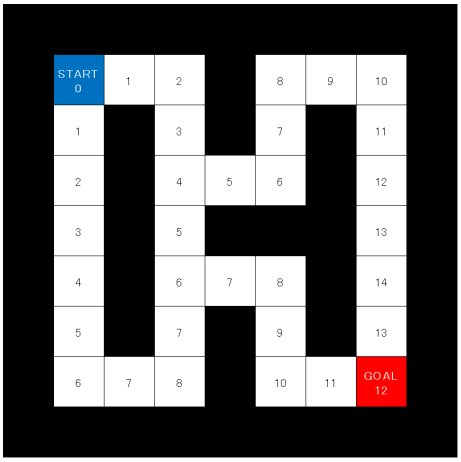

図 2.2　探索段階その 1

そして、ゴールから数字を下っていき、スタート地点まで辿っていく。数字を下っていく際に同じ数字がある場合は、どのマスを辿っても問題はない。これによってできた経路が最短経路となり、この様子を示したものが図 2.3 である。

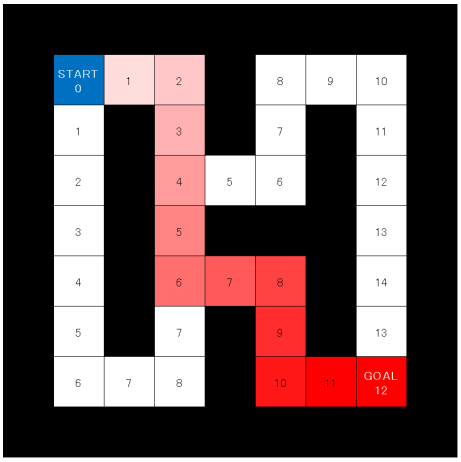

図 2.3　探索段階その 2

通過できるマスに割り振った数字の分布情報を本研究では、コストマップと定義する。

2.1.2　A*について

A*アルゴリズムは、スタートからゴールまでのコストを基に最適な経路を算出するというアルゴリズムである。第 2.1.1 項にて述べたダイクストラ法を改良したアルゴリズムである。

ダイクストラ法をどのように改良したものか、解説する。ダイクストラ法では、図 2.2 のように、移動できるマスすべてに関して数字を振っていった。対して、A*アルゴリズムでは、ゴール地点のマスに数字が振られたならば、その時点で数字を振る処理を止める。図 2.4 は数字を振る処理を停止した瞬間を表したものである。

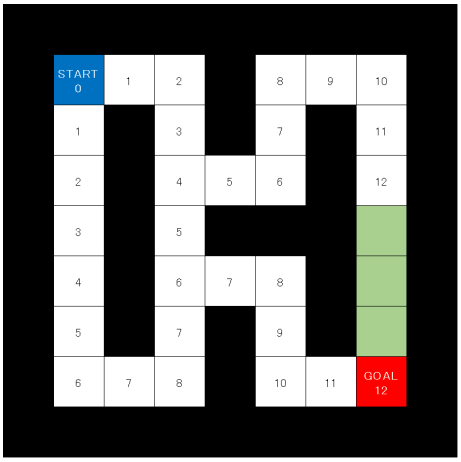

図 2.4　A*における数字の割り振り処理を停止した様子

そして、そこからはダイクストラ法のアルゴリズムと同様に、ゴールから数字を下っていくという処理を行う。

以上のように、途中で処理を止めることで従来のダイクストラ法よりも、コストマップを構築する計算の量を減らすことが実現されている。最短経路のルートとして用いないマスで、かつ、ゴール地点寄りのマスがこの A*アルゴリズムでは計算をしないで済む。

2.1.3　D*について

D*アルゴリズムは、A*アルゴリズムを動的な経路変化に対応させたものとして考案されたアルゴリズム [21] である。

矢印は、基準マスの上下左右のマスをチェックしていき、チェックしたマスに矢印が存在しなければ、基準マスを向くように設定する。はじめにゴールマスを基準マスと設定し、この処理を

行う。処理が終わった後は、チェックしたマスを新たな基準マスとし、同様の処理を行っていくことで、再帰的に矢印を設定する。一度、矢印を設定したマスは、矢印を上書きすることは無い。先ほどまで数字で表していた経路を矢印で表したものが、図 2.5 である。

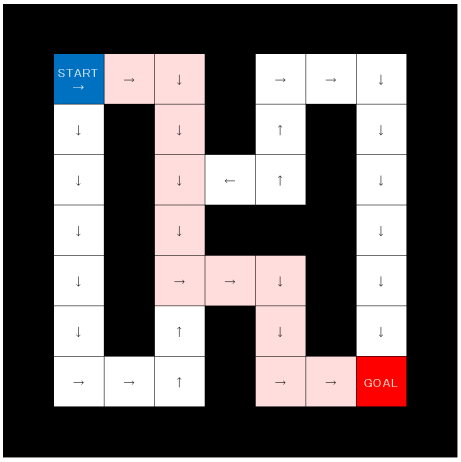

図 2.5　矢印で経路を示した様子

次に、矢印で示した最短経路上に壁を設置する。その様子が、図 2.6 である。壁を設置したことにより、通過出来ていた経路が通過できない経路へと変化する。

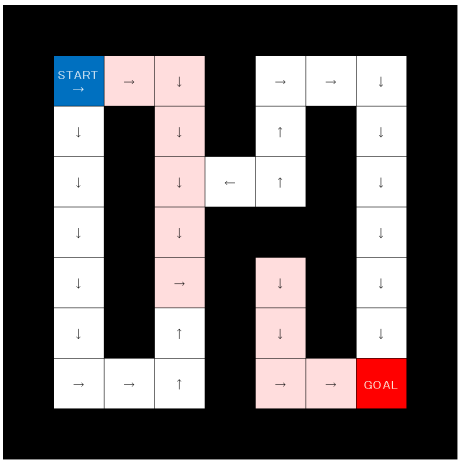

図 2.6　壁を設置した様子

ここで、ある任意地点での通路マスにおいて、そのマスが指す先のマスを親マス、さらに親マスが指す先のマスを先祖マスとし、矢印の向きによる通路のマスの接続関係を親子関係で表す。ある通路マスが壁マスに変化した際に、その他、各通路マスから見て、変化したマスが自身の先祖

に含まれるかどうかを判定する。設置した壁によって影響を受けるマスを表したものが図 2.7 である。

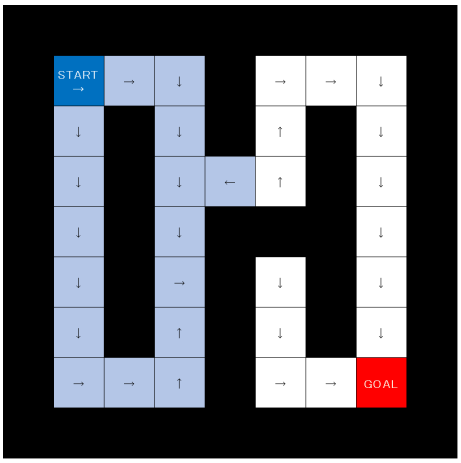

図 2.7　設置した壁の影響を受けるマスを示した様子

これらの影響マスのみを更新することで、更新頻度を減らす。これにより、マップの動的な変化に対し、効率的な再探索をするというものが D*の特徴である。影響を受けるマスのみを更新した後の矢印の様子が図 2.8 である。

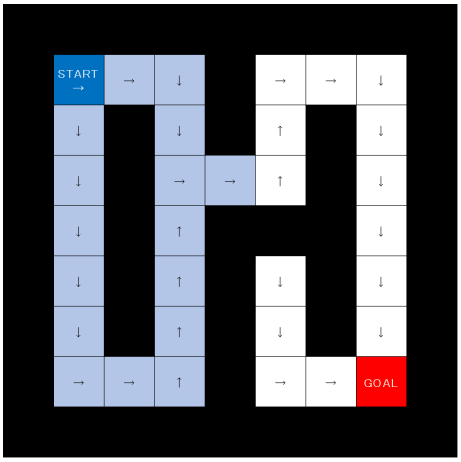

図 2.8　更新後の矢印の様子

最終的にスタートからゴールまでの経路を示したものが図 2.9 であり、図 2.5 と比較すると、経路が変化したことが分かる。

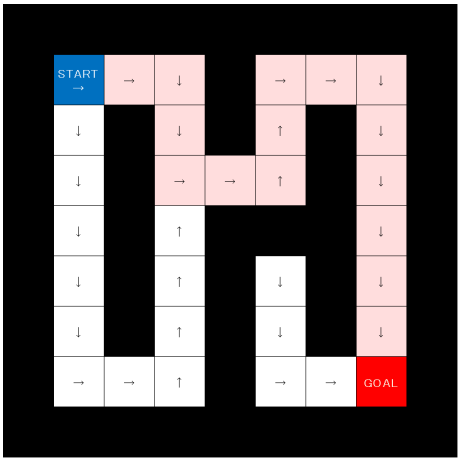

図 2.9　最終的な最短経路

影響を受けるマス以外は、更新をせずに済むマスになるため、多くの処理を行わないで済む。このアルゴリズムを使用したことによって、更新をせずに済んだマスを示したものが、図 2.10 である。

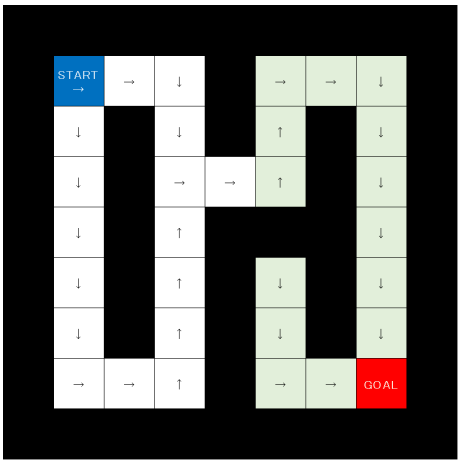

図 2.10　更新をしないで済んだマス

2.2　現状の問題

2.2.1　A*でのコストマップ再計算

問題点としてまず挙げられるのが、A*アルゴリズムのコストマップ再探索が挙げられる。壁を新たに設置すると、求めた最短経路が変化する可能性が生じて、スタートからゴールまでの経路

を再探索する必要がある。しかし、壁が設置されるたびにコストマップの再生成をしてしまうと、処理負荷がかかるとともに、時間もかかってしまう。これは、スタートからゴールまでの最短経路の長さと比例し、大きなマップになるほどこの問題は顕著になる。壁を設置したことにより、再探索が必要になったマスを示したものが、図 2.11 である。

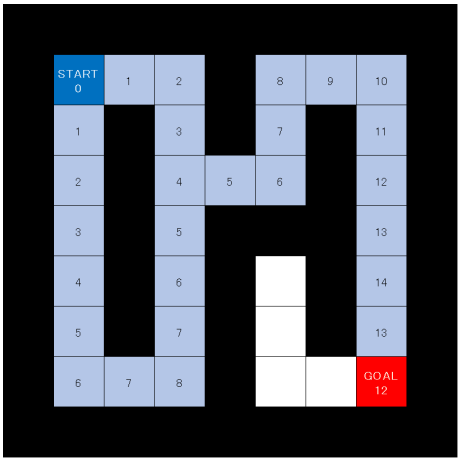

図 2.11　A*における更新処理が必要なマス

2.2.2　特定の場面における D*の冗長な処理

第 2.2.1 項にて述べたように、A*は動的な変化に対応する経路探索として非効率的である。それを解消するために考案されたのが D*アルゴリズムである。しかし、このアルゴリズムは特定の場面において冗長な処理を行うことがある。それは、更新をする必要のないマスを再更新してしまうという点である。

壁を設置していない状態の更新前と、壁を設置したことによる更新後で結果の変わらないマスが現れることがある。更新前と更新後で結果の変わらないマスを示したものが、それぞれ図 2.12 と図 2.13 である。

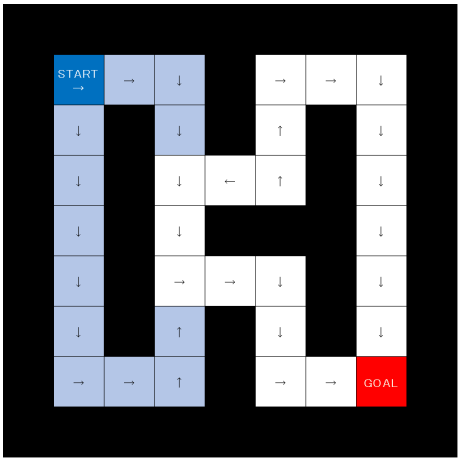

図 2.12　D*適用前のマップにおける変化のないマス

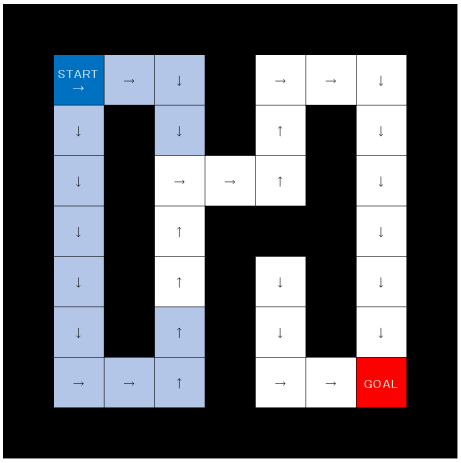

図 2.13　D*適用後のマップにおける変化のないマス

以上のことから、特定の場面において、D*アルゴリズムは冗長な処理を行う事があるといえる。

第 3 章

提案手法

本章では、経路探索について説明する。3.1 節では、提案する新たな探索手法について説明する。

本研究の手法は、最短経路数が減る場合についてのみを考慮した手法である。

3.1　提案する新たな探索手法について

提案する手法について説明する。

まず、スタートとゴールと壁と通過できるマスによって成り立つマップを用意し、スタートからゴールまでのコストマップをダイクストラ法を用いて算出する。今回例として挙げるマップは、図 3.1 である。

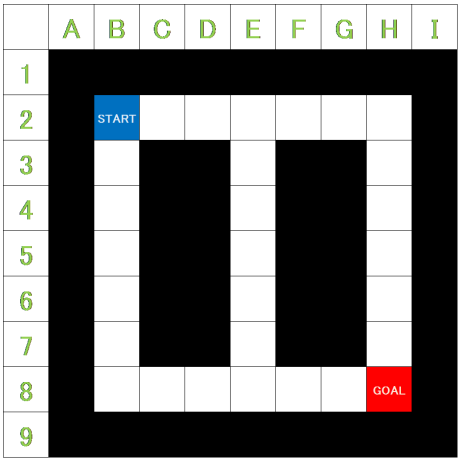

図 3.1　本手法探索対象マップ

算出結果を示したものが、図 3.2 のようになる。

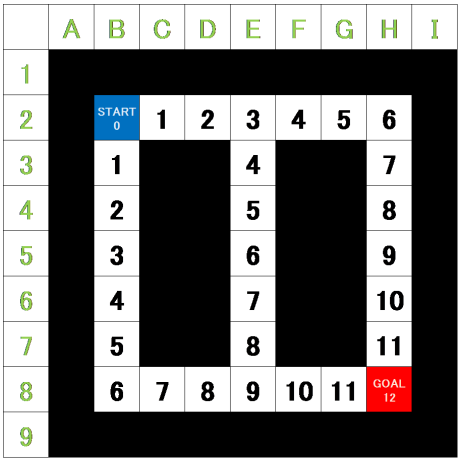

図 3.2　コストマップ算出結果

次に、コストマップを参照し、スタートからゴールまでの考えられる全ての最短経路を保存しておく。保存した経路のデータをまとめて、本研究では経路群と呼ぶ。各経路データは経路列と呼ぶ。経路列は要素としてマスを持っており、これらのマスはそのマスの位置情報とコスト情報を持っている。

経路群と経路列の模式図が、図 3.3 である。

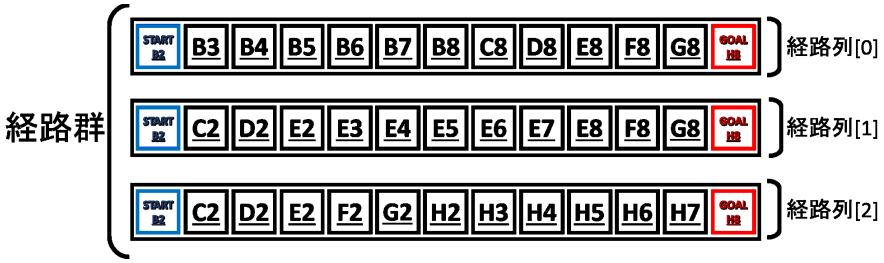

図 3.3　経路群と経路列の模式図

進路を阻まれた経路列を、保存した経路群から削除する。削除をする判定にはあらかじめ、各マスにおいて自身がどの経路列に含まれるかというデータを付加する必要がある。経路列は経路群の要素であるため、各マスにはその要素番号を保存する。

各マスにおける付加データを示したものが、図 3.4 である。

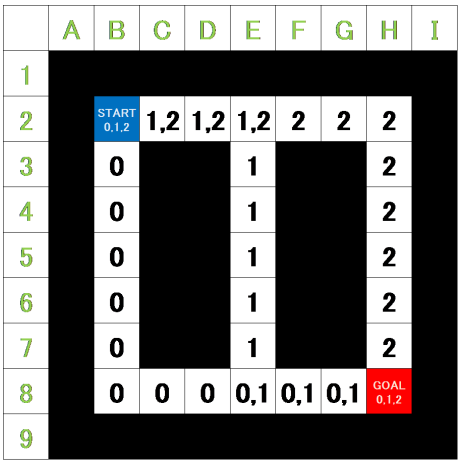

図 3.4　マスに保存される経路列

今回の図の例では、図 3.5 と図 3.6、図 3.7 である経路列が 3 つの経路群が出来る。

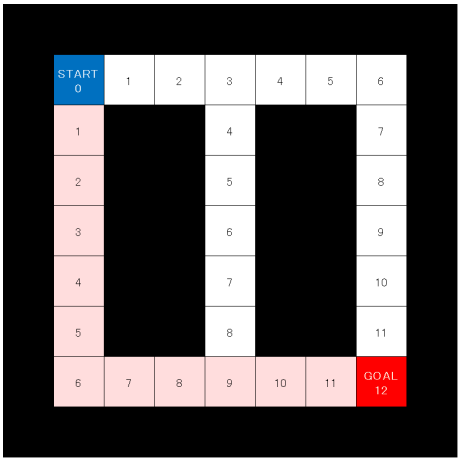

図 3.5　経路列 0 番目

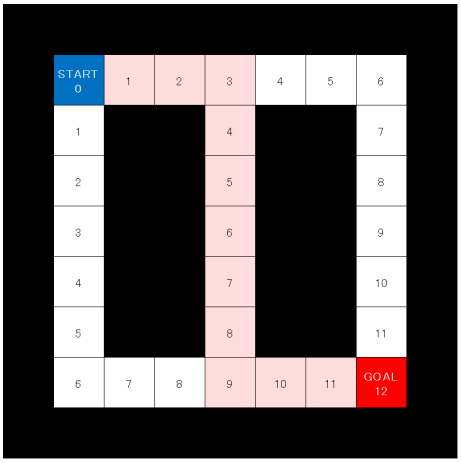

図 3.6　経路列 1 番目

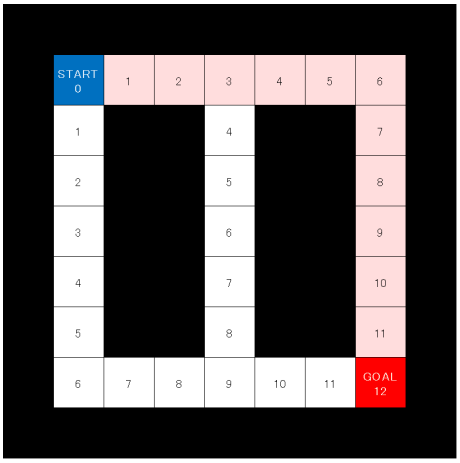

図 3.7　経路列 2 番目

ここで、経路群の要素を記すと、経路列 0 番目、経路列 1 番目、経路列 2 番目といえる。この時、いずれかの最短経路の進路を阻むように壁を設置する。今回は、経路列 1 番目の進路を妨げる。その様子が、図 3.8 である。

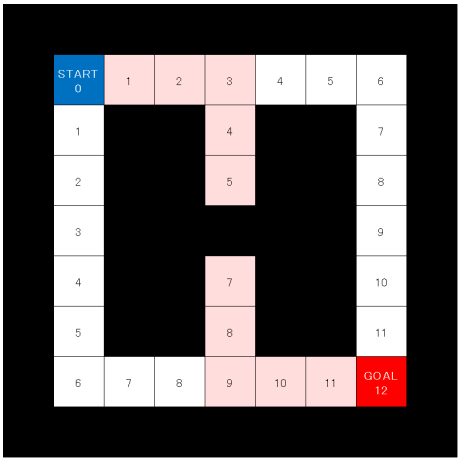

図 3.8　経路列 1 番目の進路上に障害物を設置した様子

マスが壁になれば、保存した経路データを経路群から削除する。これにより、当初の経路群から経路列を減らした新たな経路群を生成する。新たな経路群の経路列は、経路列 0 番目と経路列 2 番目の 2 つの経路列となる。あとは、この経路群から任意の経路列を抜き取ることで、最短経路を得ることが出来る。

また、この手法の特徴として、コストマップの生成をはじめのみ行い、以後、コストマップを生成する必要が無いことが挙げられる。保存した経路列が全て埋まってしまった場合に限り、コストマップの生成を行えばよい。

例では、1 つの経路列の削除を行ったが、マスに含まれた複数の経路データによって、2 つ以上の経路列の削除も対応できる。2 つの経路を防いだ様子が、図 3.9 である。

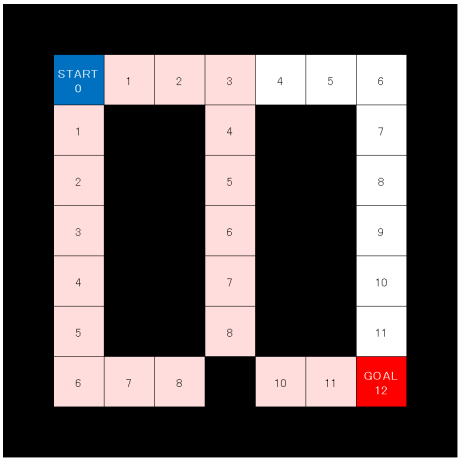

図 3.9　経路列 0 番目と経路列 1 番目の経路をともに防いだ様子

新たに生成される経路群は図 3.7 の 1 つのみの経路列となる。

第 4 章

評価と分析

本章では、実験や分析した結果について説明する。4.1 節では、実験方法について説明し、4.2 節では、実験の結果について説明する。最後に 4.3 節では、考察を述べる。

4.1　本手法と既存手法の比較

本研究の目的として、新たな経路を探索する際の処理速度の向上を目的としている。そのため、探索する際の処理速度と分散値を比較することで、実験を行う。

実験環境は以下の表 4.1 の通りである。

表 4.1　実験環境

CPU	Intel(R) Core(TM) i5-5200U CPU @ 2.20 GHz
メインメモリ	4.00GB
GPU	Intel(R) HD Graphics Family
解像度	1920 × 1080 (32 bit) (60Hz)

実験には、自身が Unity[22] で制作したプログラムを用いる。このプログラムは、A*アルゴリズムを使用した経路探索と、D*アルゴリズムを使用した経路探索、また、本研究の手法を適用した経路探索を実装している。また、このプログラムは自由に空きマスと壁マスをマス単位で設定

でき、スタートマスとゴールマスも任意の位置に変更することが可能である。図 4.1 は、実験に使用したプログラムを実行した様子である。

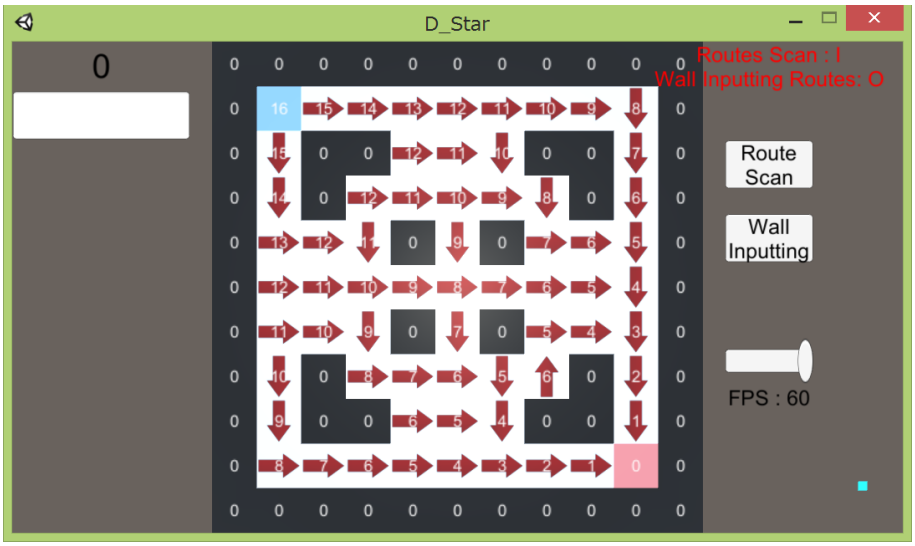

図 4.1　実装したプログラムの様子

手順としては、壁で囲われた通路のみの、マスの数が 11 × 11 のマップにスタートとゴールを任意の位置に一か所ずつ、壁を任意の位置と任意の個数、設置する。経路探索を一度行い、コストマップを生成する。そして、壁を設置して新たなコストマップを生成する。この時の壁の設置場所は空きマスのいずれかをランダムで指定する。実験はこの新たなコストマップ生成にかかる時間を A*、D*、本手法の三つの方法で計測する。計測する回数は、100 回として実験を行う。新たなコストマップを生成した後は、元のマップに戻す。計測した時間は、テキストファイルに出力する。

また、今回の実験で使用するマップはスタートからゴールまでの経路を考慮し、以下の 3 種類を用意した。

第 1 のマップは、経路に含まれる最短経路長が最少になり、最短経路数が最多となるもので、このマップを図 4.2 に示す。第 2 のマップは、経路に含まれる最短経路長が最少になり、最短経路数が中程度となるもので、このマップを図 4.3 に示す。第 3 のマップは、経路に含まれる最短経

路長が中程度あり、最短経路数も中程度となるもので、このマップを図 4.4 に示す。

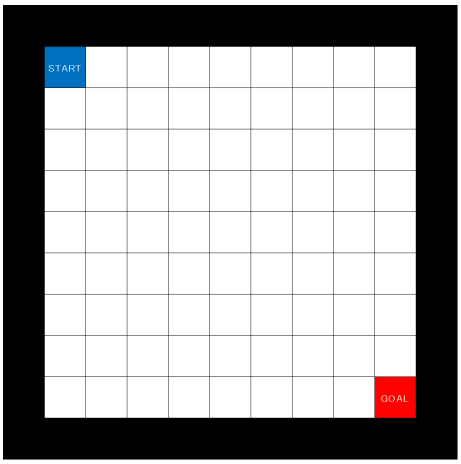

図 4.2　最短経路長が最少で経路数が最多

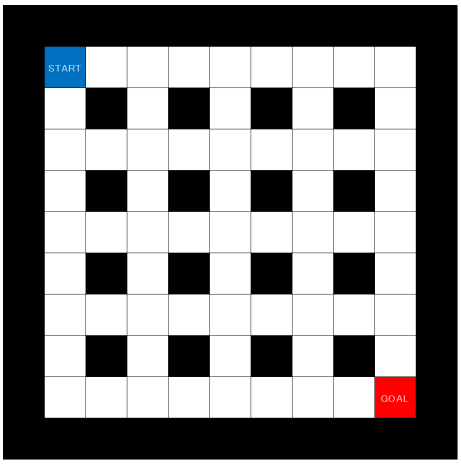

図 4.3　最短経路長が最少で経路数は中程度

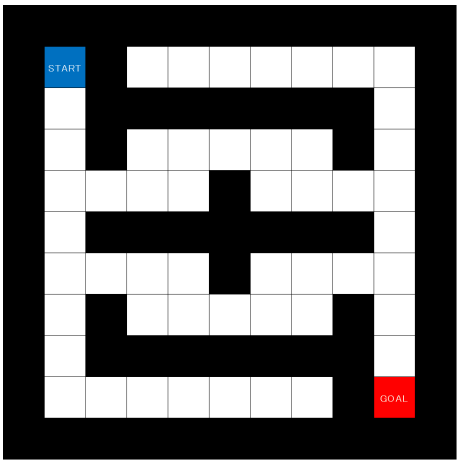

図 4.4　最短経路長も経路数も中程度

4.2　実験結果

以下に平均値の結果の表 4.2 と、その分散値の表 4.3 を示す。数値は小数第 3 位まで有効とする。

表 4.2　平均処理速度 (ミリ秒)

	A*	D*	本手法
最短経路長最少で経路数最多	4.815	11.764	78.144
最短経路長最少で経路数中程度	5.445	7.321	4.088
最短経路長中程度で経路数中程度	3.650	2.371	2.969

表 4.3　処理速度の分散

	A*	D*	本手法
最短経路長最少で経路数最多	11.930	55.028	4767.895
最短経路長最少で経路数中程度	10.249	22.319	14.269
最短経路長中程度で経路数中程度	8.741	3.512	1.298

数値は僅かではあるが、処理速度が向上したマップが見受けられる。最短経路長が最少で経路数が中程度なマップと、最短経路長が中程度で経路数が中程度なマップが D*よりも速い結果を出している。

4.3　考察

図 4.2 の、最短経路長が最少で経路数が最多のマップの結果を見ると、A*、D*、本手法の順に遅くなっていることが分かる。また、事前準備に 1 分ほど使用している。他のマップでは、数秒で完了するこの事前準備に時間がかかるということは、本手法は向いていないことがよく分かる。このことから、経路数が増えてしまうような例には、A*を用いた方がより効率的に探索できる事が分かる。分散の値をみても明らかで、本手法がこのパターンに向いていないことが断言できる。

次に、図 4.3 の、最短経路長が最少で経路数は中程度あるマップの結果を見ると、本手法が最も早く処理できたことが分かる。また、この場合における D*は A*よりも遅いことが分かり、A*が如何に優秀なアルゴリズムかが伺える。さらに、本手法の分散の値を見ると、D*アルゴリズムより、バラつきが少ないことが分かる。

最後に、図 4.4 の、最短経路長と経路数共に中程度あるマップの結果を見ると、速い順に、D*、本手法、A*であることが分かる。D*の強みの出るパターンがこれに該当することが分かった。分散値をみると、本手法が最もバラつきが少ないといえる。

以上のことから、A*、D*、本手法には得意なパターンと苦手なパターンがあることが分かる。本手法が得意なパターンは、最短経路長が少なく、経路数が中程度あるパターンにおいて効果があるといえる。

本手法は、FPS ゲームにおける市街戦のような複数の障害物が多く配置されるような入り組んだステージでは有効であるが、荒野のような障害物の少ない開けたステージでは、有効ではないといえる。

第 5 章

まとめ

従来の経路探索アルゴリズムである、A*と D*を本手法と共に比較してみた。比較したことで、マップのパターンごとの得手不得手が各々にあることが分かった。本研究で提案した手法は、最短経路長と経路数が最少とは言わないがある程度少ないマップにおいて有効であることが判明した。また、従来の A*アルゴリズムは最短経路長が少なく、経路数が多い場合に強力であり、D*アルゴリズムは、最短経路長が多く、経路数が最少の時に非常に強力であることが分かった。

以上のことから、マップのパターンによってアルゴリズムの使い分けを行う事で、より高速で、動的な変化に頑健な処理が行えると考えた。

今回の手法では、経路が壁に阻まれることによる変化のみを扱った。展望として、壁が除去されるという変化についても対応可能な手法を考える必要がある。また、どのようなパターンのマップにおいても、効率的な処理が行えるように、マップのパターンによる、アルゴリズムの使い分けも急務である。

一方で、本研究の手法が適用できない例外的な場面が存在する。本手法が適用できない場面は、以下の 3 パターンがある。

- 最短経路ではないマスに壁を設置した場合
- 全ての最短経路に共通するマスに壁を設置した場合
- スタートからゴールまでの経路がなくなる場合

まず、最短経路ではないマスに壁を設置した場合だが、これは、どの最短経路に対しても進路を防ぐことはないため、経路群が変化することもない。そのため、コストマップの再生成は行わない。処理時間も皆無である。

次に、全ての最短経路に共通するマスに壁を設置した場合だが、これは、すべての経路群要素が削除されてしまうことを意味している。そのため、経路群を新たに生成するために、コストマップを再生成する必要がある。コストマップを生成してから、さらに新たな経路群を生成する必要があるため、この処理には時間を要してしまう可能性が十分にある。

最後に、スタートからゴールまでの経路がなくなる場合だが、これは経路探索自体が行えないため、壁をなくし、再び探索する必要がある。

謝辞

本論文を執筆するにあたり、ご指導いただきました渡辺先生、阿部先生に心より感謝いたします。

また、様々な相談に親身に応じてくださった研究室のメンバー、友人たちにこの場を借りて深く感謝いたします。

誠にありがとうございました。

参考文献

[1] David M. Bourg, Glenn Seemann. ゲーム開発者のための AI 入門. O'Reilly Japan, Japan, 2005.

[2] 独立行政法人 工業所有権情報・研修館. 平成 16 年度 特許流通支援チャート カーナビ経路探索技術. http://www.inpit.go.jp/blob/katsuyo/pdf/chart/fdenki22.pdf. 参照:2018.7.15.

[3] 高橋一幸、Temsiririrkkul Sila、池田心. ローグライクゲームの研究用ルール提案とモンテカルロ法の適用. ゲームプログラミングワークショップ 2017 論文集, Vol. 2017, pp. 19–25, 2017.

[4] 藤井叙人、佐藤祐一、中嶌洋輔、若間弘典、風井浩志、片寄晴弘. 生物学的制約の導入による「人間らしい」振る舞いを伴うゲーム ai の自律的獲得. ゲームプログラミングワークショップ 2013 論文集, Vol. 2013, pp. 73–80, 2013.

[5] 2018 Unity Technologies. Publication 2018.1-X. ナビメッシュエージェント. https://docs.unity3d.com/jp/current/Manual/class-NavMeshAgent.html. 参照:2018.7.8.

[6] Epic Games. NavMesh コンテンツサンプル. http://api.unrealengine.com/JPN/Resources/ContentExamples/NavMesh/. 参照:2018.7.15.

[7] pafuhana1213. ぼっちプログラマのメモ. http://pafuhana1213.hatenablog.com/entry/2014/06/08/190825. 参照:2018.7.15.

[8] 櫻場悠之、須貝康雄. 利便性を考慮した力学系に基づく複数経路探索. 情報処理学会 第 80 回全国大会講演論文集, Vol. 2018, pp. 309–310, 2018.

[9] 鈴木一平、今井桂子. 車両型ロボットの経路生成に関する一手法の提案. 情報処理学会研究報告 アルゴリズム（AL）, Vol. 1, pp. AL–99, 2005.

[10] 高松秀行、築地立家. 自動走行ロボットによる曲がり角検出アルゴリズムの研究. 情報処理学会 第 77 回全国大会講演論文集, Vol. 2015, pp. 647–648, 2015.

[11] 福田隼馬、阿部和規、藤井秀樹、山田和典、吉村忍. 大規模マルチエージェント高通流シミュレーションのための階層的経路探索手法. 情報処理学会論文誌, Vol. 59, No. 7, pp. 1435–1444, 2018.

[12] 丸三徳、関口隆昭、林新、天谷真一. 車両のレーン変更を考慮した経路探索方式. 情報処理学会論文誌 コンシューマ・デバイス＆システム（CDS）, Vol. 8, No. 2, pp. 66–73, 2018.

[13] 古次なぎ、阿部真也、山本佳世子. 粘菌アルゴリズムによる避難経路の導出. 情報処理学会第 80 回全国大会, Vol. 2018, No. 1, pp. 425–426, 2018.

[14] 梅木寿人、中村優吾、藤本まなと、水本旭洋、諏訪博彦、荒川豊、安本慶一. 災害時の混雑情報を考慮した避難所決定手法の提案. 情報処理学会 研究報告モバイルコンピューティングとパーベイシブシステム（MBL）, Vol. 2018-MBL-86, No. 20, pp. 1–8, 2018.

[15] Dijkstra, E.W. A note on two problems in connexion with graphics. *Numerische Mathematik, 1*, Vol. S, pp. 269–271, 1959.

[16] コルテ, B.、フィーゲン, J. 組み合わせ最適化：理論とアルゴリズム. シュプリンガー・ジャパン, Japan, 2009.

[17] Jorudan. ジョルダン 乗換案内. https://www.jorudan.co.jp/norikae/. 参照:2018.7.16.

[18] NAVITIME JAPAN. NAVITIME 乗換案内. https://www.navitime.co.jp/transfer/. 参照:2018.7.16.

[19] KIM PALLISTER, 川西裕幸, 中本浩. *GAME PROGRAMMING Gems5*. 株式会社 ボーンデジタル, Japan, 2006.

[20] nitoyon(にとよん). 経路探索アルゴリズムの「ダイクストラ法」と「A*」をビジュアライズしてみた. http://tech.nitoyon.com/ja/blog/2010/01/26/dijkstra-aster-visualize/. 参照:2018.7.9.

[21] Howie Choset. Robotic Motion Planning:A* and D* Search. https://www.cs.cmu.edu/~motionplanning/lecture/AppH-astar-dstar_howie.pdf. 参照:2018.7.9.

[22] 2018 Unity Technologies. Unity 2018.2、リリース. https://unity3d.com/jp. 参照:2018.7.15.