修士論文　　2019 年度

コストマップの局所的な再設定による
最短経路再探索の高速化手法に関する研究

東京工科大学大学院

バイオ・情報メディア研究科

メディアサイエンス専攻

津川 巧

修士論文　　2019 年度

コストマップの局所的な再設定による
最短経路再探索の高速化手法に関する研究

指導教員　　渡辺 大地 准教授

東 京 工 科 大 学 大 学 院

バ イ オ ・ 情 報 メ デ ィ ア 研 究 科

メ デ ィ ア サ イ エ ン ス 専 攻

津川 巧

論 文 の 要 旨

論文題目	コストマップの局所的な再設定による 最短経路再探索の高速化手法に関する研究
執筆者氏名	津川 巧
指導教員	渡辺 大地 准教授
キーワード	経路探索、ゲーム AI、再探索、 接続情報、マルチスレッド

[要旨]

経路探索は、ロボット AI やゲーム AI などに用いられる。経路探索アルゴリズムとして有名なものにダイクストラ法や A*アルゴリズムがある。マップが目まぐるしく変化するようなゲーム作品などにおける最短経路の再探索を高速化することは重要であるといえる。

しかし、従来のアルゴリズムでは再探索において膨大な量のマップを探索しなくてはならないため、非効率的である。マップの変化に対応して、最短経路の再探索を行うアルゴリズムに D*アルゴリズムがある。しかし、このアルゴリズムでは、より最短である新たな最短経路が出来る場合に対応していない。本研究ではマップの変化時における新たに最短経路が出来る場合に着目した最短経路の高速な再探索手法について、コストマップの局所的な再設定による経路再探索手法を提案する。

全域分析フェーズ、局所分析フェーズ、経路検出フェーズの 3 つのフェーズを利用する。全域分析フェーズでマップ全体を分析し、マップ全体において正確な分析結果を得る。局所分析フェーズでは、全域分析フェーズで得た分析結果を限定的に分析することで、効率的に範囲内の情報を再設定することが出来る。高速に処理をすることが出来るが、変化した箇所から離れた情報は元のままで不正確な状況である。一度のマップ変更の場合は局所分析フェーズを行う事で厳密な経路を得ることができるが、さらにマップが変更された場合は再度全域分析フェーズを行わなければならない。そこで、マルチスレッドにより全域分析フェーズを行う事で、正確な情報の更新を待たずに経路探索を行う機能を実現した。

複数のマップを対象に経路の再探索時の処理速度を計測した結果、従来のアルゴリズムよりも高速に再探索を行う事が出来た。これにより、再探索を即時的に行い、新たな経路を得ることが可能になった。しかしながら、従来の経路探索ではスタート地点とゴール地点のどちらか一方が動く場合には経路探索を行う事は阻まれないが、本手法では、スタート地点とゴール地点からの情報を利用しているため、一方が動いてしまう場合では経路探索が行えない点がある。

A b s t r a c t

Title	Study on speed-up method of shortest path re-search by local reconfiguration of cost map
Author	Takumi Tsugawa
Advisor	Taichi Watanabe
Key Words	PathSearch, GameAI, Retry, Connect Info, Multithread

[**summary**]

Route search is used for robot AI and game AI. Dijkstra's algorithm and A * algorithm are well known as route search algorithms. It is important to speed up the search for the shortest path in game works where the map changes rapidly.

However, the conventional algorithm is inefficient because it has to search a huge amount of maps in the re-search. The D * algorithm is an algorithm that re-searches for the shortest path in response to changes in the map. However, this algorithm does not support the case where a new shortest path is created. In this study, we propose a new route search method based on the localization of the cost map.

Uses three phases: the whole analysis phase, the local analysis phase, and the route detection phase. The entire map is analyzed in the whole area analysis phase, and accurate analysis results are obtained for the entire map. In the local analysis phase, the information within the range can be efficiently reset by analyzing the analysis results obtained in the whole area analysis phase in a limited manner. Although the processing can be performed at high speed, the information distant from the changed portion is an inaccurate situation as it is. In the case of a single map change, a strict route can be obtained by performing the local analysis phase, but if the map is further changed, the entire analysis phase must be performed again. Therefore, by performing the whole area analysis phase using multi-threads, a function of performing a route search without waiting for accurate information update has been realized.

As a result of measuring the processing speed when re-searching the route for a plurality of maps, the re-search was performed faster than the conventional algorithm. This makes it possible to immediately perform a re-search and obtain a new route. However, in the conventional route search, if either the start point or the goal point moves, it is not prevented to perform the route search.However, since this method uses information from the start point and the goal point, However, if one of them moves, the route search cannot be performed.

目 次

図 目 次

表 目 次

第 1 章

はじめに

1.1　背景と目的

経路探索は、目的地点への移動経路を算出するものである。ロボット AI やゲーム AI、交通シミュレーション [1] や災害時の経路シミュレーション [2] などに用いられる。近年のゲームに多く使用されているものとして、ナビゲーションメッシュがある。これは経路探索のためのポリゴンメッシュのことで、ゲーム内オブジェクトを考慮し、土地の形状に合わせて生成する。

経路探索は、ノードと呼ばれる頂点によって構成されるグラフを対象に行われる。電車の路線図を例に挙げると各駅がノードに対応し、路線図が 1 つのグラフであるといえる。図 1.1 は、JR 東日本による八王子駅周辺の路線図である。

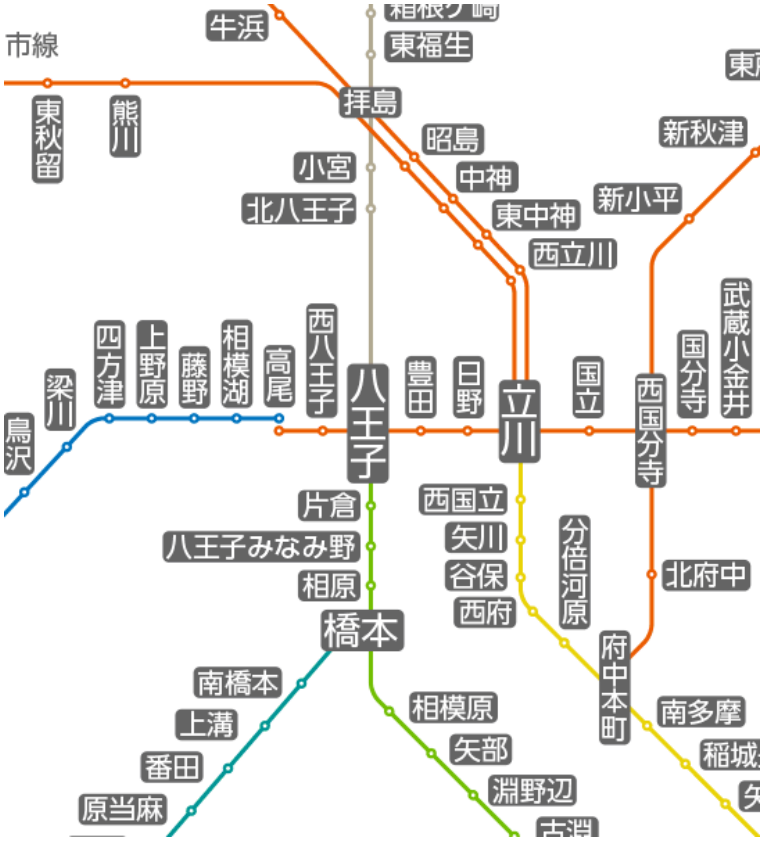

図 1.1　八王子周辺の路線図

https://www.jreast.co.jp/map/

経路探索の基本ともいえるアルゴリズムの 1 つに、Dijkstra[3] が提唱したダイクストラ法があ

る。このアルゴリズムは、ノード間における正のコストから 2 点間の最適な経路を算出するものである。乗換案内にも使用されており [4]、駅間における運賃や所要時間をコストに対応させることで適切な経路を探索する。一般的に経路探索では、初めに探索をするグラフを分析し、その分析した情報を基に探索を行う。グラフに動的な変形が起こった際の経路の再探索では、ダイクストラ法のようなアルゴリズムで再度分析、探索をしていては処理時間が増加し非効率的である。

MINECRAFT[5] やボンバーマン [6]、地球防衛軍 [7] のようなマップが目まぐるしく変化するようなゲーム作品などでは、経路探索の再探索の高速化は、インタラクティブ性を重視するため非常に重要であるといえる。MINECRAFT では、フィールド上に自由にものを設置や削除ができる。ボンバーマンでは、爆弾を設置することで壁を破壊することが可能である。地球防衛軍では、自分の攻撃や敵の攻撃によってフィールドに設置してあるビルや家が消し飛び、自由に行動できる範囲が増加する。このように、ゲーム作品において自由に動き回れるフィールドが変化することがしばしばおこる。

経路探索には様々な種類があり、昔から多くの研究 [8][9] [10] がされている。サッカーシミュレーションの経路決定の研究 [11][12] [13] も多く行われている。チェスや将棋のような完全情報ゲームと呼ばれる分野では、ゲーム木探索を行う事で AI が作成されており、これを利用した研究 [14][15] [16] が多くされている。モンテカルロ法を利用したヒューリスティックな木探索のことをモンテカルロ木探索という。これを利用した研究 [17][18] [19] も多く行われている。不確定不完全情報ゲームにおける最適行動の決定に関する研究 [20][21][22] も行われている。

2 点間最短経路問題を解くために Peter ら [23] は、A*アルゴリズムを提案した。このアルゴリズムは、ダイクストラ法の探索時における経路探索時の冗長な部分を省略するために考案されたアルゴリズムである。A*アルゴリズムは多くの場合、探索対象ノードとゴールまでのユークリッド距離をヒューリスティック関数とし、冗長な探索を省略している。冗長な探索を省略するアプローチによって経路探索の高速化が可能である。

経路の再探索の手法として、幾つか研究がなされている。その中に Stenz[24] が提唱した D*アルゴリズムがある。D*アルゴリズムは、変化の有ったノードに関連の有るノードのみを更新するアルゴリズムである。この手法は、経路変化時の中でも経路が妨げられる場合に考慮されている。関連の有るノードは、各ノードにあらかじめ設定されている進路情報を参照することで判別を行う。これにより、効率的な再探索を行う事が可能になる。このアルゴリズムは経路が塞がれてしまい、最適であった経路が無くなる場合の再探索に対し利点が発揮できる。しかしながら、より最短である新たな最短経路が出来る場合に利点を発揮できない。

Goldberg ら [25] が提唱した ALT アルゴリズムは、まず探索対象となるマップ上に幾つかの目印を設置する。マップに変化があった際は、設置した目印を参考に A*アルゴリズムを用いて再探索を行うというものである。しかしながら、指標にするランドマークをどこにいくつ設定するかといったランドマークの諸設定が難しいといった点がある。

Wagner ら [26] が提唱した Geometric Containers では、マップのレイアウトを幾何学的に分析し、幾何学情報を抽出する。円や多角形であるなどの抽出した幾何学情報を基にすることで、ダイクストラアルゴリズムの探索スペースを大幅に減らすことが出来るというものである。しかしながら、高速化が可能ではあるものの、大きな短縮には至らない点がある。

Berrettini ら [27] が提唱した Arc Flags は、Geometric Containers の改良版アルゴリズムであり、分析にかかる時間をより高速にしたものである。しかしながら、経路探索自体のパフォーマンスが若干低下してしまうという点がある。

Geisberger ら [28] が提唱した CH は、Contraction Hierarchies の略であり、各ノードに対して重要度を付与することで階層分けを行うものである。マップが変化した際は階層を参照することで効率的に再探索が行えるというアルゴリズムである。しかしながら、時間や距離といった経路探索における指標の拡張性が低く、また、階層分けによる重要度が低い部分の探索には大きな処理時間を要してしまう。

Deling ら [29] が提唱した CRP は、CH を改良したアルゴリズムで、CH の弱点であった経路探索における指標の拡張性が高いという点で強みがあるアルゴリズムである。だが、再探索の速度が必ずしも高速とは言えないという点がある。

ここで、マップ変化に対応した手法である D*アルゴリズムでは、より最短である新たな最短経路が出来る場合に対応していない。他の手法においては、再探索速度の高速化を最優先としておらず、グラフの分析にかかる時間の高速化やデータ量の削減を目的にしてる。いずれも最適な経路の即時的な算出を行うアプローチではないため、即時的な経路算出に特化したアルゴリズムが無い。

本研究の目的は、マップが変化した際の経路の再探索を高速に行う事である。また、その中でも経路が新たに出現するマップ変化に対して、高速に再探索を行う手法を提案する。経路の再探索を高速に行う事で、高速な処理を要求するゲームに利用できると強力であると考える。

本手法では、ゴールとスタートの 2 つのノードまでの距離を利用する。従来の経路探索アルゴリズムでは、ゴールまでの距離のみで経路探索を行うところを本手法ではスタートまでの距離を利用することで、局所的な再探索が可能になる。また、経路探索の為に行う分析を全域分析と局所分析の 2 つに分けることで、即時的な経路の算出と 2 回目以降の経路の再探索に頑健に対応できる。全域分析と局所分析の 2 つの分析処理は、マルチスレッド処理により、即時的な経路算出と 2 回目以降の経路の頑健な再探索を実現した。

特徴の違うマップで従来のアルゴリズムであるダイクストラ法と A*アルゴリズムと本手法でマップ変化時の経路再探索速度を比較した。その結果、本手法の経路算出までの時間が最も速いことが分かった。次回以降の再探索のためにマップの情報を正確なものにする処理時間は従来のアルゴリズムと同様に必要であるが、即時的な経路再探索を実現することが出来た。

1.2　論文構成

本論文は全 4 章にて構成する。2 章では本手法について述べ、3 章では実験について述べる。そして 4 章にてまとめを述べる。

第 2 章

提案手法

2.1 表記定義

本研究におけるグラフ各部の数式表現は、基本的には文献 [30] に準拠するものとし、以下のように定める。

- 本論文では頂点のことをノードと呼称する。また、ゴールノードのことを P_g、スタートノードのことを P_s と表記する。図 2.1 で、ゴールノードとスタートノードを示す。
- ノード間を結ぶものを辺という。また、辺は長さを持つ。図 2.2 で、辺を示す。
- 辺が P_i と P_j を結んでいるとき、P_i と P_j は隣接するという。図 2.3 で、隣接するノードと隣接していないノードを示す。P_1 と P_2 は隣接しているが、P_2 と P_4 は隣接していない関係にある。
- 経路 W は、順に伝っていたノードの列と定義し、経由するノードを要素として、

$$W = \{P_0, P_1, \cdots, P_n\} \tag{2.1}$$

と表記する。図 2.4 で、経路の一例を示す。

$$W = \{P_s, P_3, P_1, P_2, P_g\} \tag{2.2}$$

と表す。

- 各ノードにおける隣接するノードの中で、最もゴールまでの距離が小さいノードを次ノードとする。同様に、最もスタートまでの距離が小さいノードを前ノードとする。
- P_0 を始点、P_n を終点とする経路の内、長さが最短となる経路の長さを

$$D(P_0, P_n) \tag{2.3}$$

と表現し、これを P_0 と P_n の距離と呼ぶ。

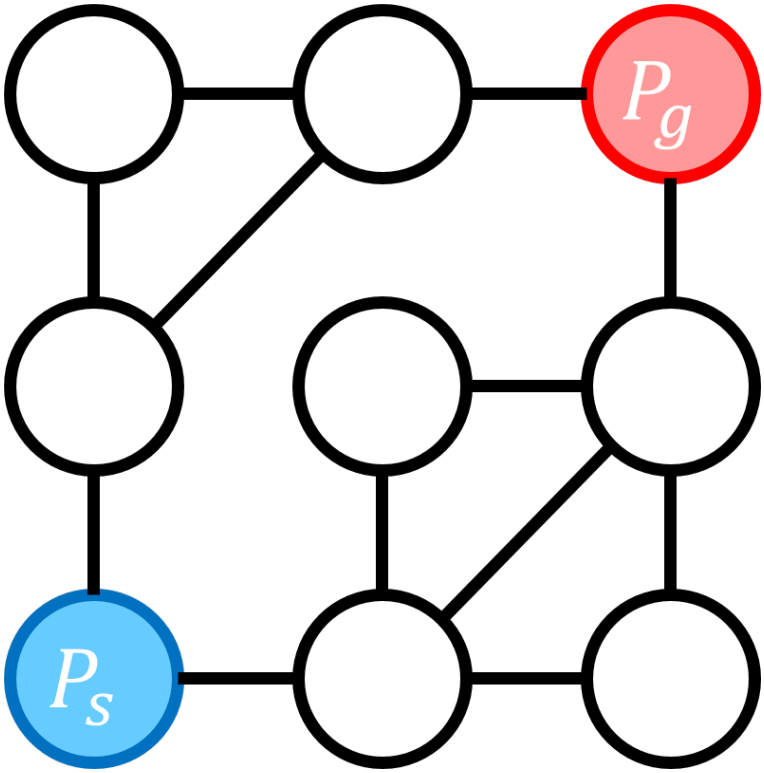

図 2.1　ゴールノードとスタートノード

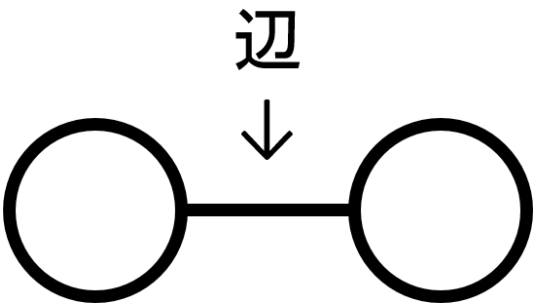

図 2.2　辺の一例

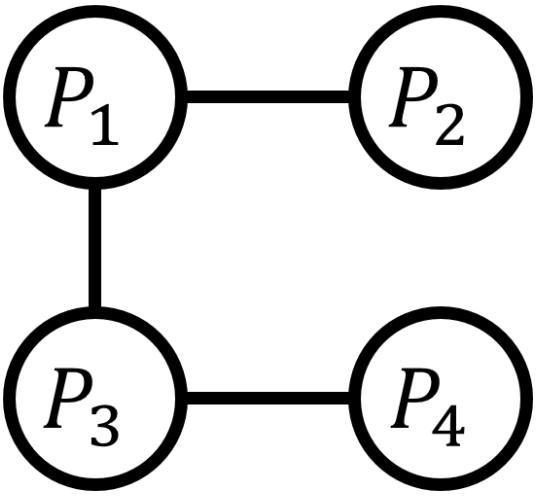

図 2.3　隣接するノードの一例

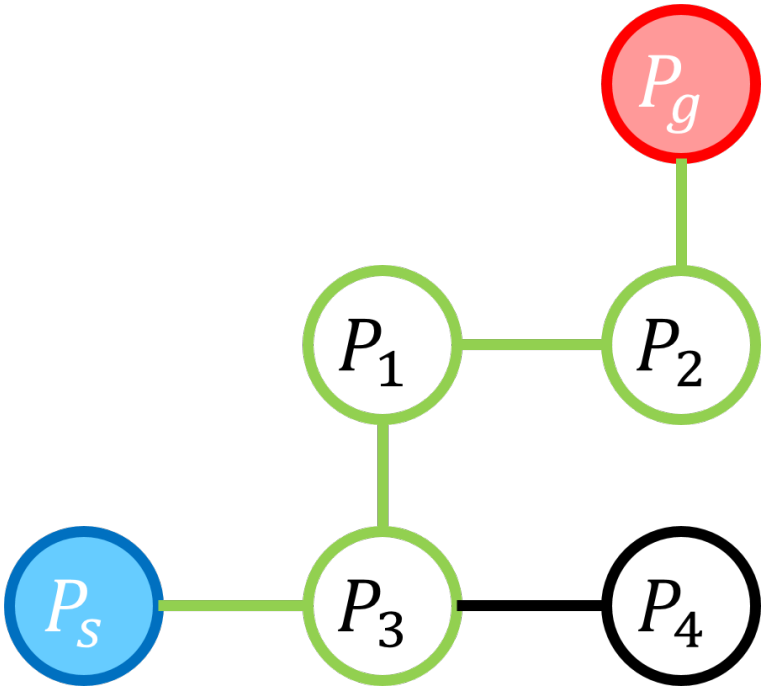

図 2.4　経路の一例

2.2　提案手法の流れ

本手法では、ノード間の距離を算出した値をコストと呼称する。このコスト算出を、全域分析フェーズ、局所分析フェーズで行う。また、算出したコストを利用し経路を検出する、経路検出

フェーズの計 3 つのフェーズで行う。全域分析フェーズではマップ全体を分析し、全ノードの完全コストマップを生成する。完全コストマップについては次節で詳しく解説する。マップに変化があった際に、局所分析フェーズを実行する。ここでは完全コストマップの限定的な範囲を分析することで、効率的に範囲内のノードの距離のみを算出し、局所コストマップを生成する。局所コストマップの算出は完全コストマップに比べ高速に行えるが、変化した箇所から離れたノードのコストは変化前のままで、不正確な状態となる。しかしながら、厳密なコストマップではないが概ね良い経路を算出することが可能な状況であり、コストマップが完全になるまではこの状態で経路探索を行っていく。

その後、全域分析フェーズによって全てのノードの正確なコストを算出するが、この全域分析フェーズをマルチスレッドで行う事により、コストマップの再計算の完了を待たずに経路探索を行う機能と完全なコストマップの算出を両立する。

2.2.1 全域分析フェーズ

経路検出の説明のため、あらかじめ全域分析フェーズでの説明が必要であるため全域分析フェーズを先に説明する。全域分析フェーズでは、本手法における経路探索に必要な情報を全ノードに設定する。ここで生成するコストマップのことを完全コストマップと呼称する。ダイクストラ法を用いて、以下の 4 つの情報を設定する。

- ゴールまでの距離
- スタートまでの距離
- 次ノード情報
- 前ノード情報

ここで、次ノード情報とは、ノードからゴールへの最短経路上での次ノードとなり得るノード群の

ことである。また、前ノード情報も同様に算出する。以下に全域分析フェーズの流れを示す。図 2.5 で、探索対象マップを示す。このマップに対し全域分析フェーズを行う事で、4 つの情報を設定する。図 2.6 で、ゴールまでの距離と次ノード情報を示す。また、図 2.7 で、スタートまでの距離と前ノード情報を示す。図におけるゴールノードの持つ前ノード情報はゴールノードに隣接するノードで、スタートノードの持つ次ノード情報はスタートノードに隣接するノードとなる。

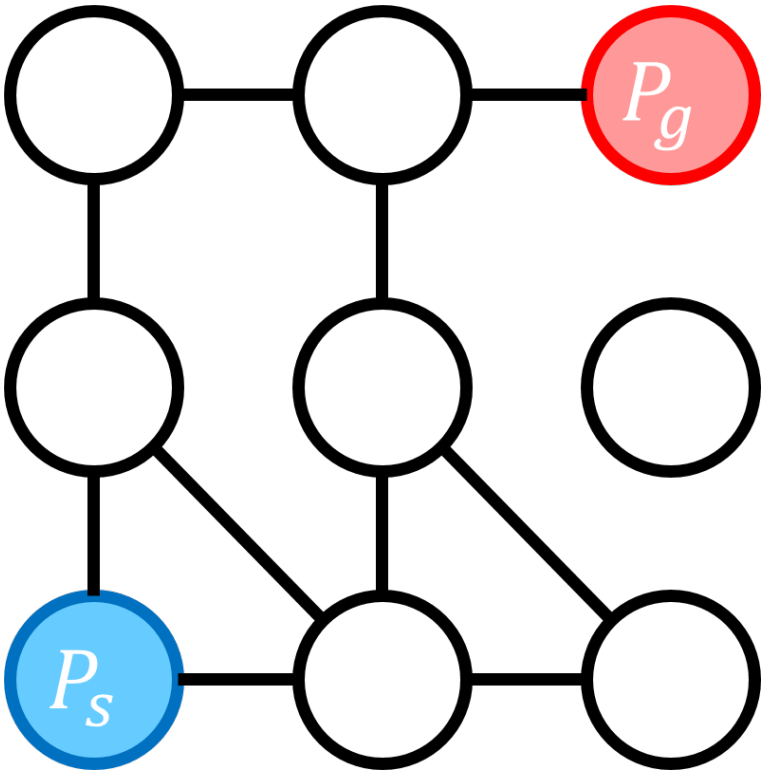

図 2.5　探索対象マップ

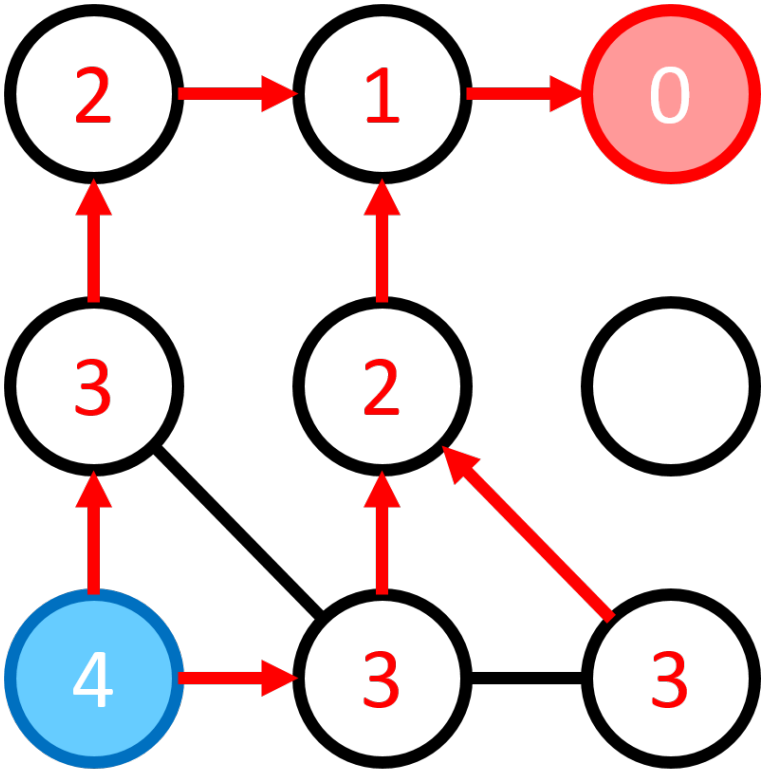

図 2.6　ゴールまでの距離と次ノード情報

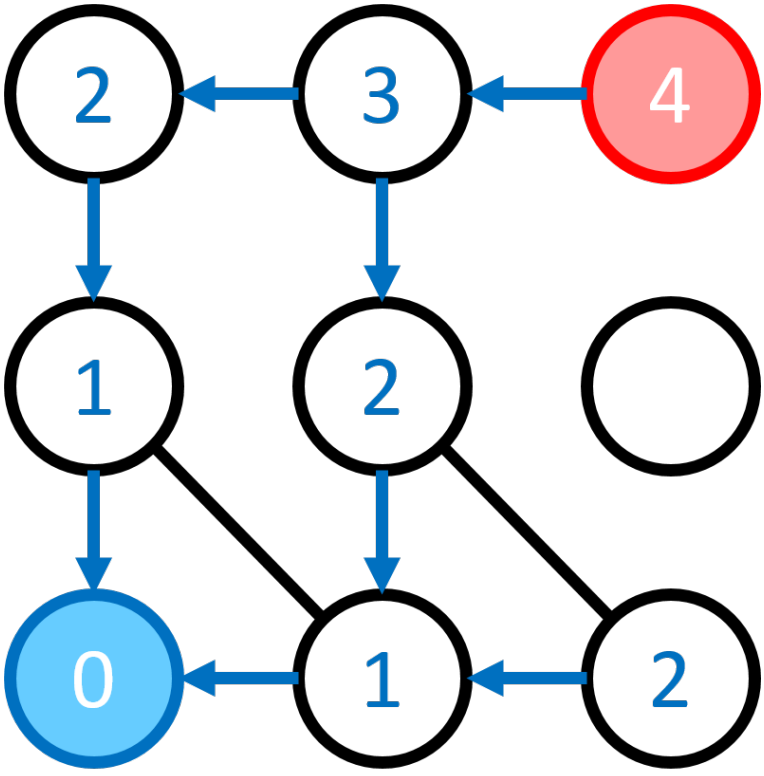

図 2.7　スタートまでの距離と前ノード情報

2.2.2　局所分析フェーズ

マップ変化によって新たな経路が出現した場合、局所分析フェーズを行う。本フェーズでは、前ノード情報を利用し、特定のノードの次ノード情報を再設定する。再設定するノードの候補が全域分析に比べて少ない為、局所的な再構築が可能である。

新たな経路が出現した最初の処理として、有効ノードのゴールまでの距離と次ノード情報を設定する。ここで、有効ノードとは、接続する辺の数が 0 のノードがマップの変化により接続する辺が現れて、辺の数が 0 ではなくなったノードのことを指す。図 2.8 は、有効ノードが出現する例を簡易的に表したものである。図 2.9 は、全域分析フェーズでの図 2.5 における有効ノードが出現する例を示したものである。本論文では、有効ノードを P_r と表記する。有効ノードの隣接するノードの内、最もゴールノードに近いノードを P_q と表記する。ゴールまでの距離は、式 (2.4) により決定する。それと同時に、有効ノードの次ノードを P_q とする。図 2.10 は、有効ノードに設定したゴールまでの距離と次ノードを示したものである。

$$D(P_r, P_g) = D(P_q, P_g) + D(P_q, P_r) \tag{2.4}$$

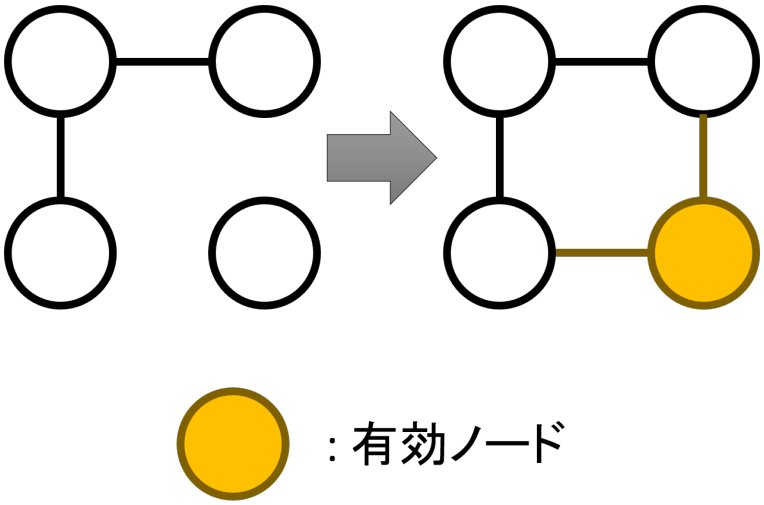

図 2.8　有効ノードの簡易的な出現例

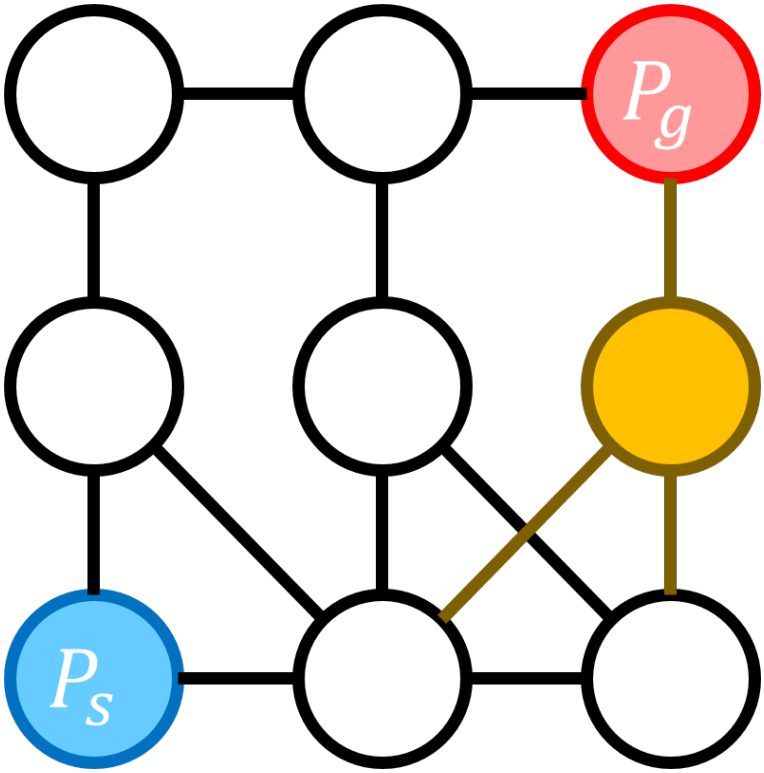

図 2.9　探索対象マップにおける有効ノードの出現例

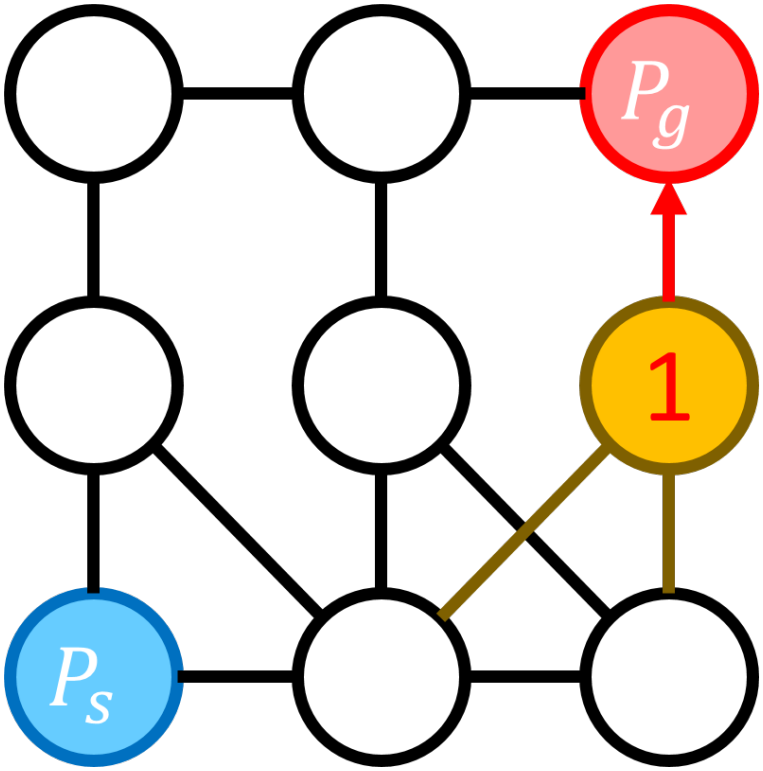

図 2.10　探索対象マップにおける有効ノードの情報を設定した様子

次に、スタート近接ノードのゴールまでの距離を再設定する。ここで、スタート近接ノードとは、有効ノードに隣接するノードの内、最もスタートノードに近いノードのことを指す。図 2.11 は、各頂点間の辺が均一なコストを持つ場合のグラフにおける、スタート近接ノードを簡易的に表したものである。図 2.12 は、全域分析フェーズでの図 2.5 におけるスタート近接ノードを示したものである。本論文では、スタート近接ノードを P_t と表記する。スタート近接ノードのゴールまでの距離は、式 (2.5) により決定する。それと同時に、スタート近接ノードの次ノードを P_r とする。図 2.13 は、スタート近接ノードに設定したゴールまでの距離と次ノードを示したものである。

$$L(P_t, P_g) = L(P_r, P_g) + L(P_r, P_t) \tag{2.5}$$

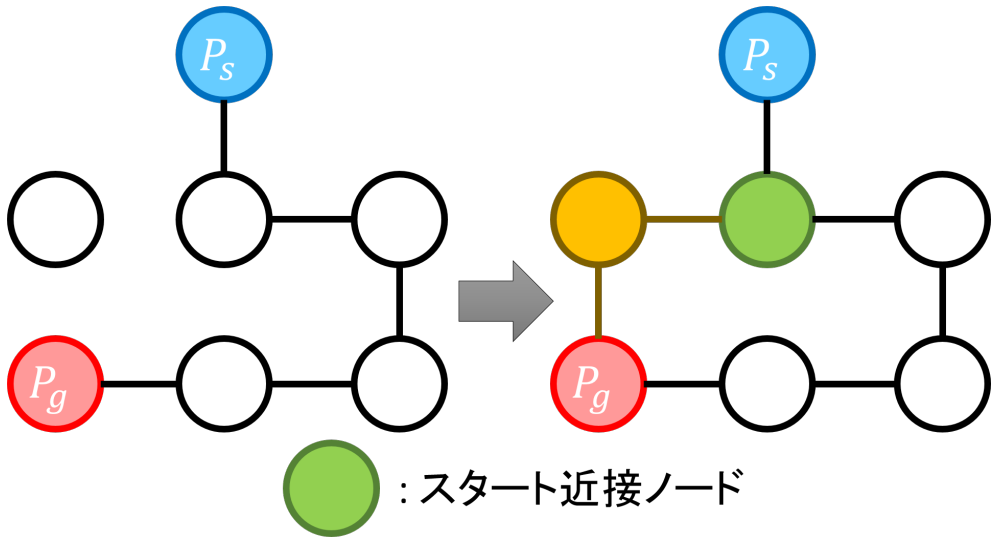

図 2.11　スタート近接ノードの簡易的な例

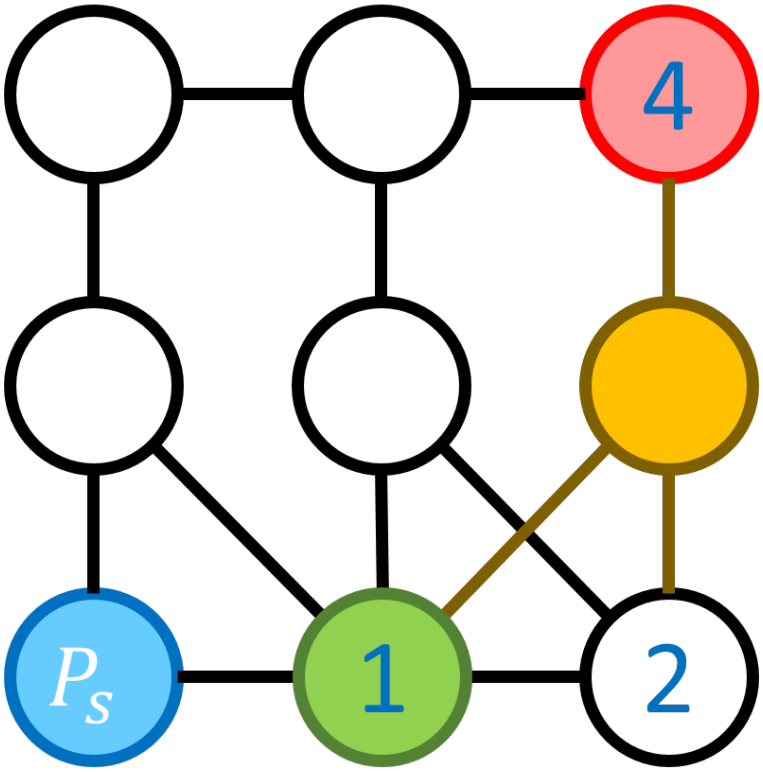

図 2.12　探索対象マップにおけるスタート近接ノードの例

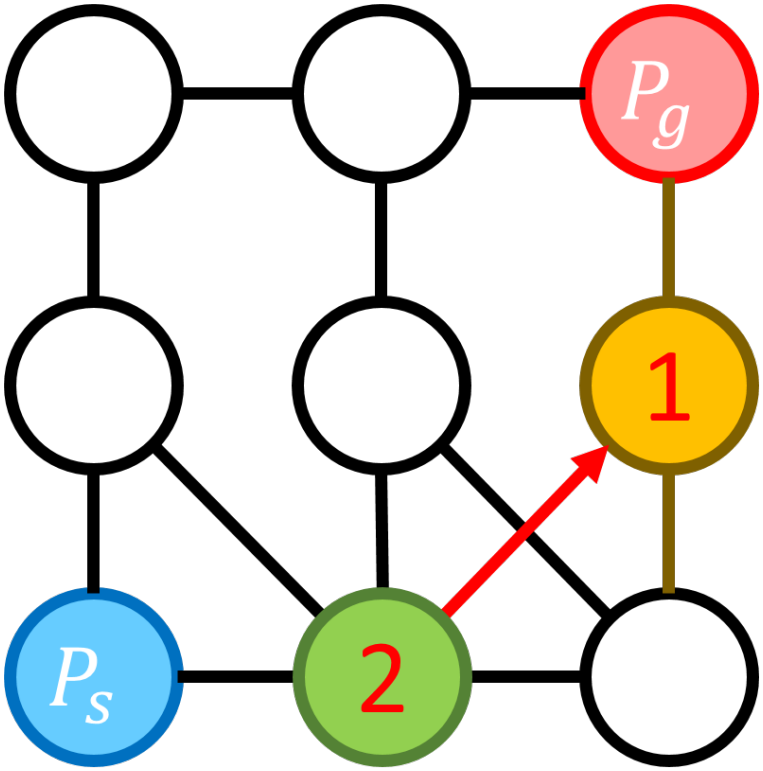

図 2.13　探索対象マップにおけるスタート近接ノードの情報を再設定した様子

次に、次ノード情報を再設定するノードをスタート近接ノードから見つける。スタート近接ノードは、全域分析フェーズによって前ノード情報を持っているため、この情報をスタートノードまで参照していくことで再設定すべきノードが見つかる。スタート近接ノードからスタートノードまで順番に次ノード情報を再設定していく。図 2.14 は、スタート近接ノードの持つ前ノード情報をスタートノードまで参照した様子を示したものである。また、図 2.15 は、参照したノードの次ノード情報を更新する様子を示したものである。

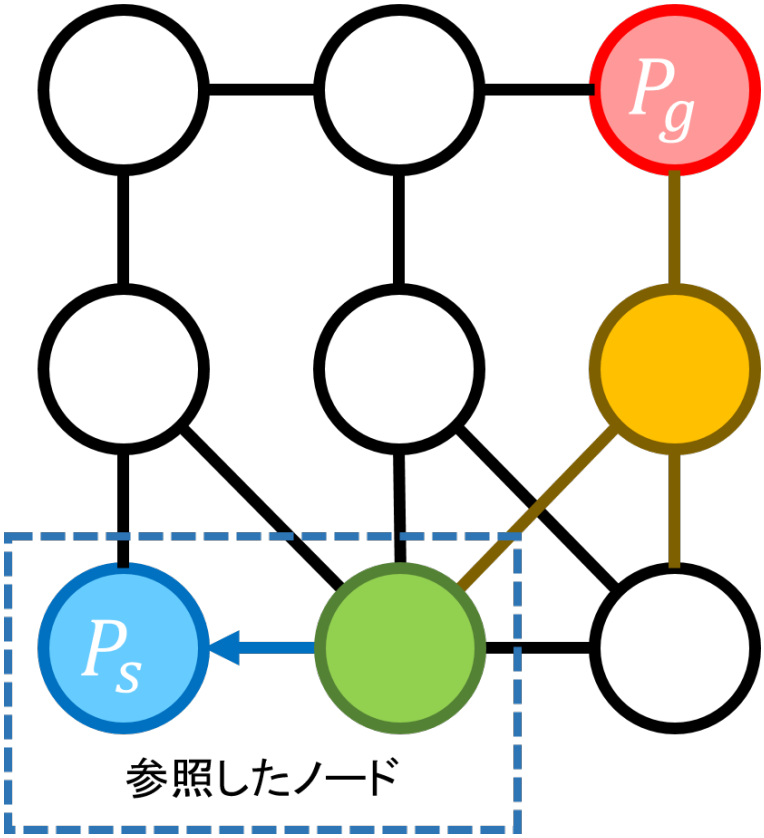

図 2.14　探索対象マップにおけるスタートノードへの参照の様子

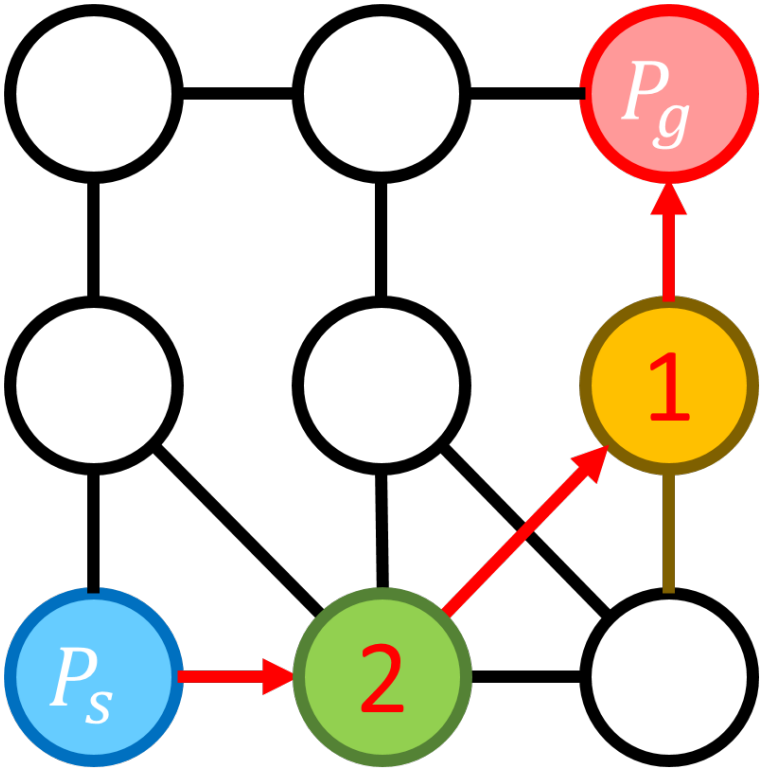

図 2.15　探索対象マップにおける次ノード更新の様子

2.2.3　経路検出フェーズ

最後のフェーズとして、スタートからゴールまでの最短経路を求める経路検出フェーズを説明する。経路検出フェーズでは、スタートノードにおける次ノード情報を参照するだけで新たな最短経路を得ることが出来る。新たな経路が出来た際の局所分析フェーズにて次ノード情報を局所的に再設定していたため、このフェーズをもって最短経路の再探索が適切に行われる。

例にしたグラフよりも大規模なグラフを例にして示したものが、図 2.16 から図 2.25 である。図 2.16 は、大規模なグラフの一例である。図 2.17 は、ゴールまでの距離と次ノード情報を示したものであり、図 2.18 は、スタートまでの距離と前ノード情報を示したものである。図 2.19 は、有効ノードが現れた様子を示したものである。また、図 2.20 は、有効ノードのゴールまでの距離と次ノード情報を設定したものである。図 2.21 は、スタート近接ノードを示したものである。また、

図 2.22 は、スタート近接ノードのゴールまでの距離と次ノード情報を再設定したものである。図 2.23 は、スタート近接ノードから前ノード情報を順に参照したものである。図 2.24 は、参照したノードを更新した様子を示したものである。図 2.25 は、経路出現フェーズにより発見された経路を示したものである。

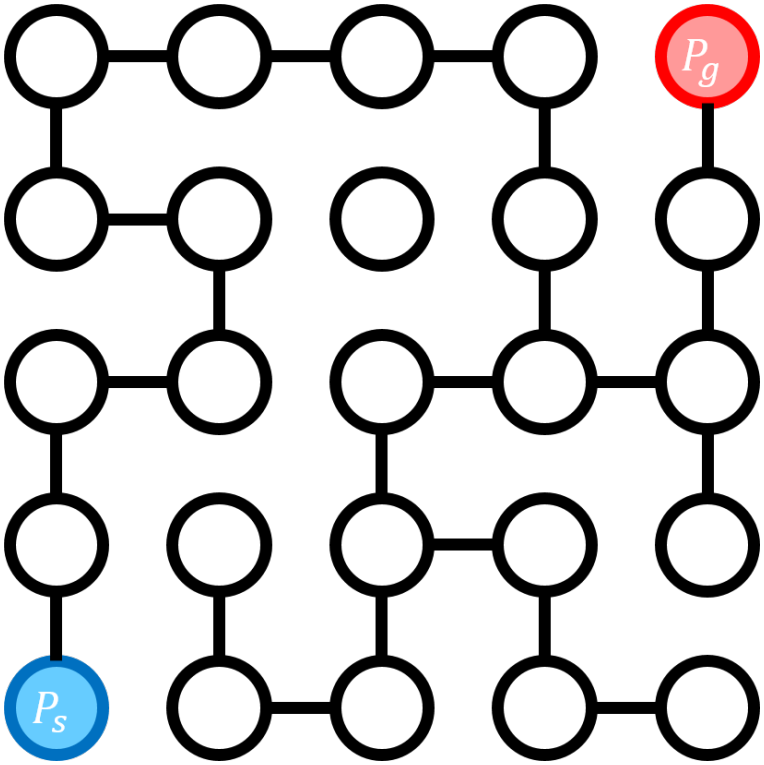

図 2.16　大規模なグラフの一例

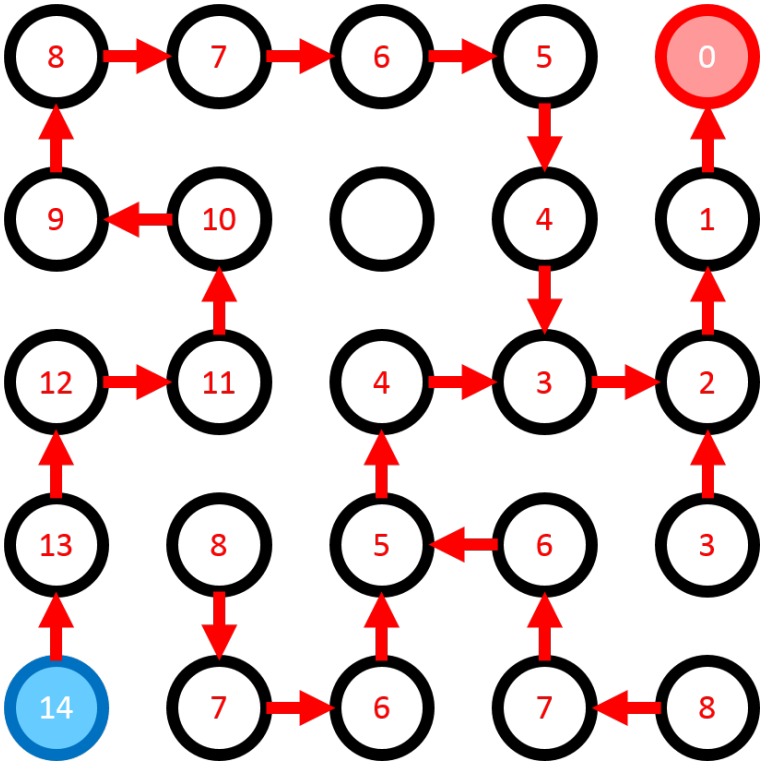

図 2.17　大規模なグラフのゴールまでの距離と次ノード情報

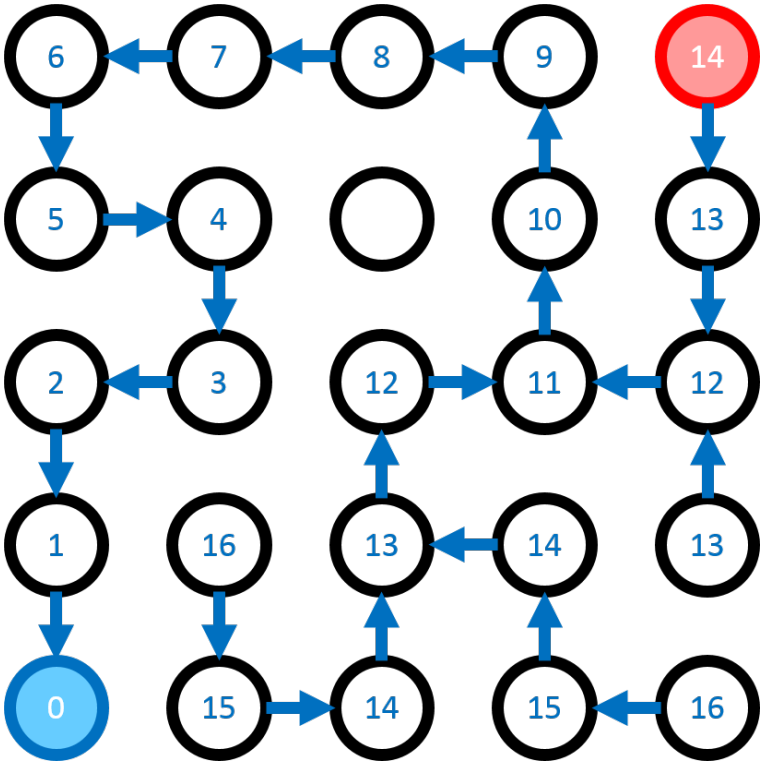

図 2.18　大規模なグラフのスタートまでの距離と前ノード情報

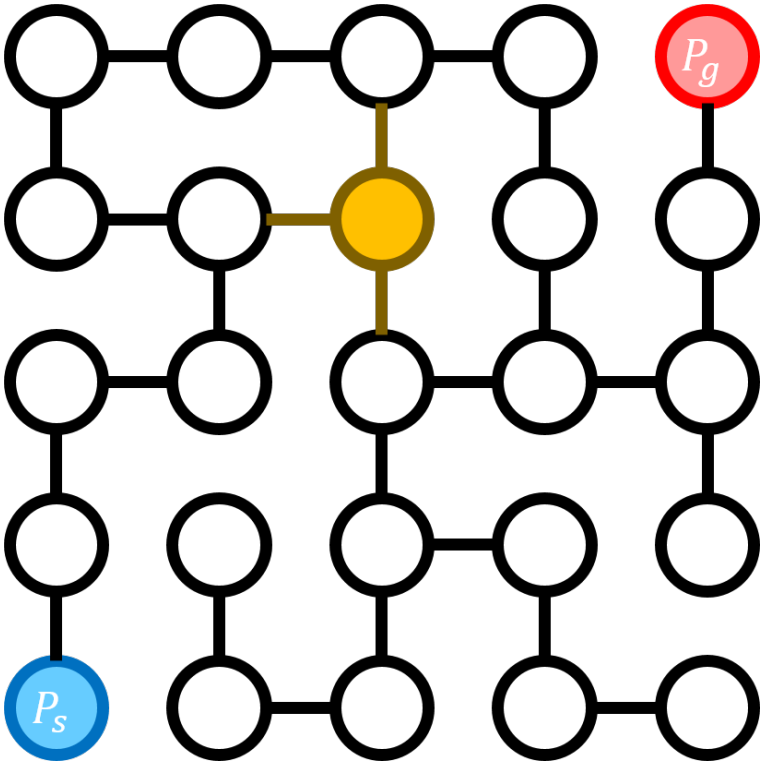

図 2.19　大規模なグラフの有効ノード

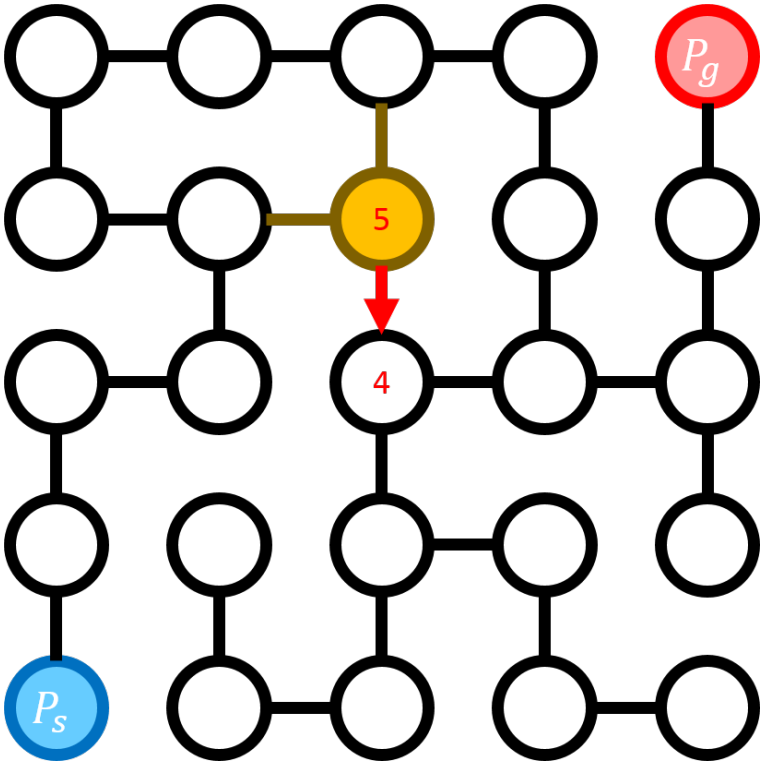

図 2.20　大規模なグラフの有効ノードの情報を設定した様子

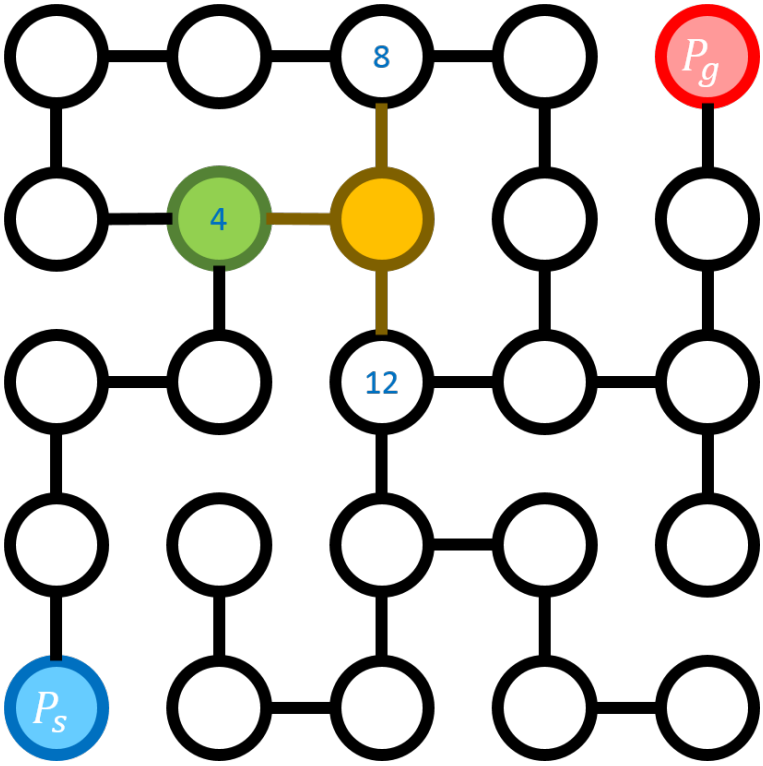

図 2.21　大規模なグラフのスタート近接ノード

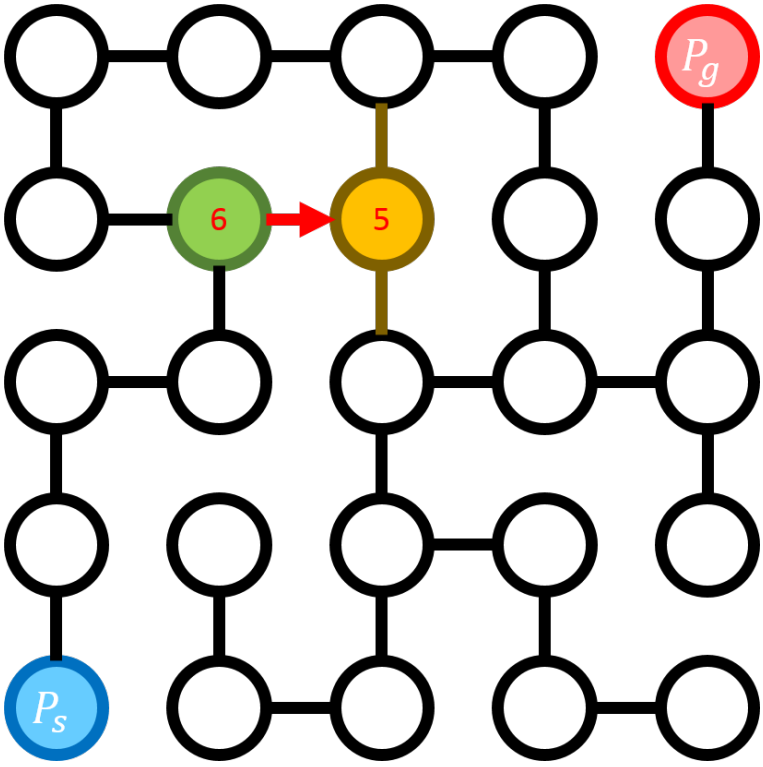

図 2.22　大規模なグラフのスタート近接ノードの情報を再設定した様子

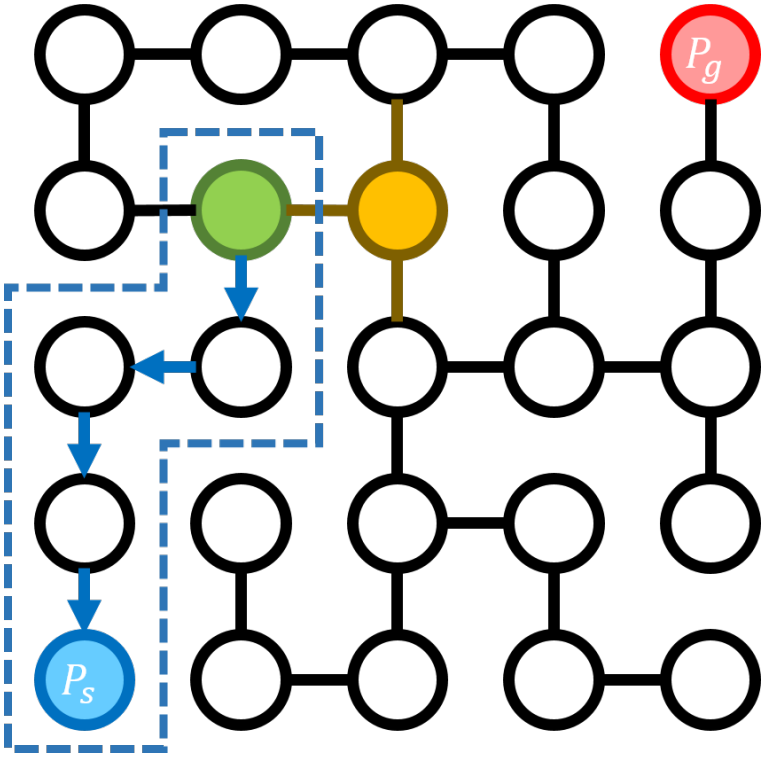

図 2.23　大規模なグラフのスタート近接ノードから参照した様子

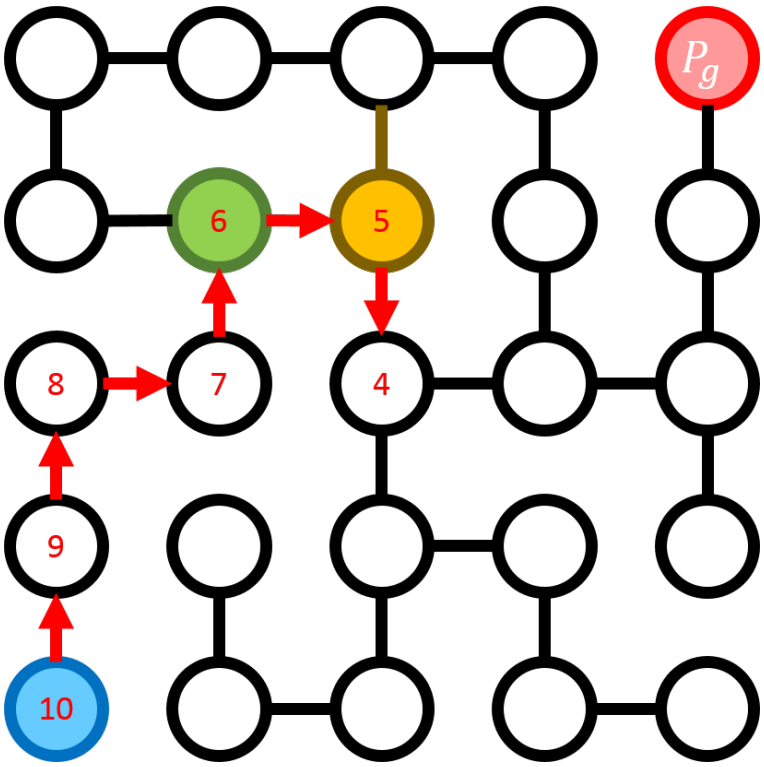

図 2.24　大規模なグラフの参照したノードを更新した様子

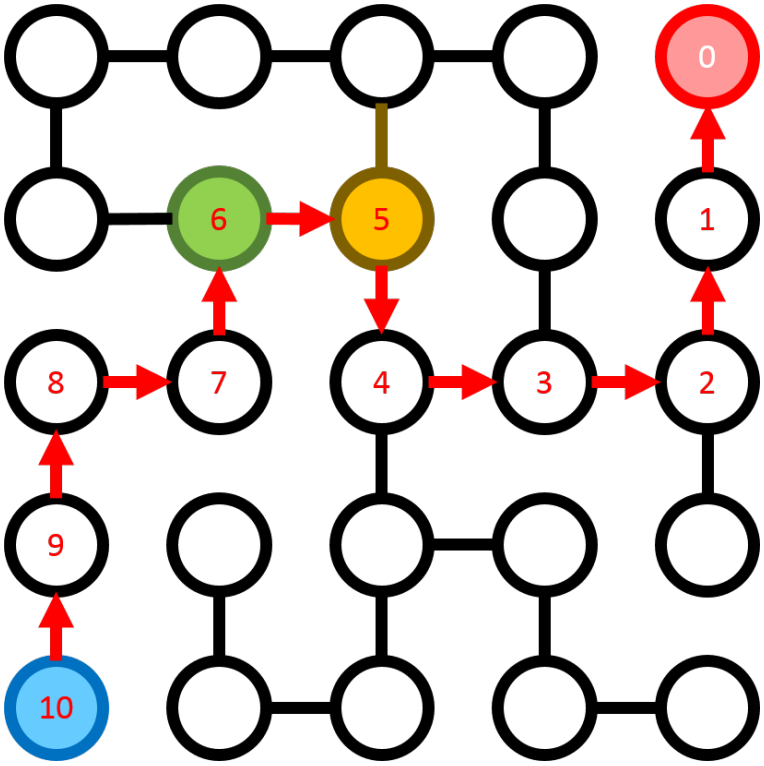

図 2.25　大規模なグラフの最短経路を発見した様子

2.2.4　再探索が必要ではない場合の判定

有効ノードが現れることで再探索を行ったとしても、新たな最短経路は発生しない場合がある。再探索を行う必要が無い場合に再探索を行う事は無駄な処理である。そこで、マップ変化時の新たな経路が最短経路になる場合とならない場合を判定する必要がある。全域分析フェーズと局所分析フェーズを行う前に、この判定を行う。この判定で利用する情報は局所分析フェーズでも利用する。

まず、再探索を必要としない場合は 2 パターンある。それぞれのパターンを示したものが図 2.26 と図 2.27 である。

- 新たに通過できるノードが出来たが、そのノードを通過しては迂回ルートになってしまう場合

- 新たに通過できるノードが出来たが、そのノードを通過する場合もしない場合も最短経路になる場合

図 2.26 は、迂回ルートになる場合を示したものである。また、図 2.27 は、最短経路の距離が変わらない場合を示したものである。

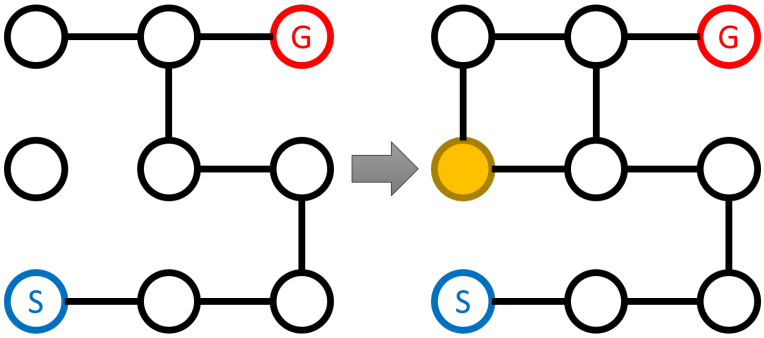

図 2.26　迂回ルートになる場合

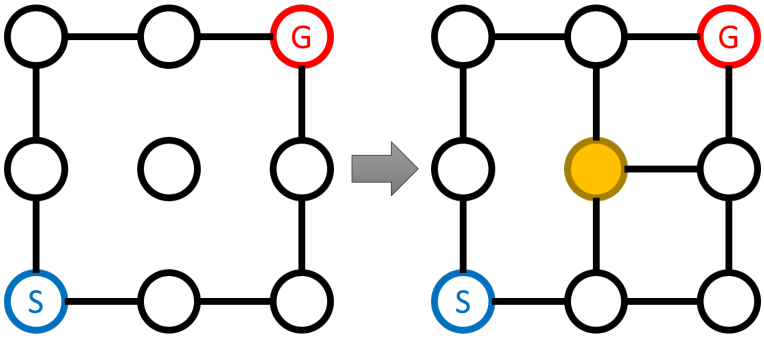

図 2.27　最短経路の距離が変わらない場合

以上の場合を判定するために、再設定した $D(P_t, P_g)$ と全域分析フェーズで算出済みの $D(P_t, P_s)$ とマップ変化前のスタートノードからゴールノードまでの最短経路の長さの 3 つを利用することで判定する。P_s から P_g までの最短経路を W_p とすると、判定式が式 (2.6) である。

$$D(P_t, P_g) + D(P_r, P_g) \geq D(P_r, P_t) \tag{2.6}$$

式 (2.6) を満たす場合は、最短経路とならないと判定する。その際、スタート近接ノードの前

ノード情報を参照する処理を省略する。これにより、最短経路とならない場合の冗長な再設定処理を省くことが出来る。

2.2.5　複数の接続先を保持している場合の参照方法

各ノードが複数のノードを次ノード情報または前ノード情報として保持している場合は、接続先を参照する際にいずれか 1 つの接続先を参照すればよい。これにより、さらに局所的なゴールノードへの接続情報の再設定が可能になる。図 2.28 で、次ノード情報と前ノード情報を複数保持しているノードがあるグラフを示す。図 2.29 は、次ノード情報を示すものであり、図 2.30 は、前ノード情報を示すものである。

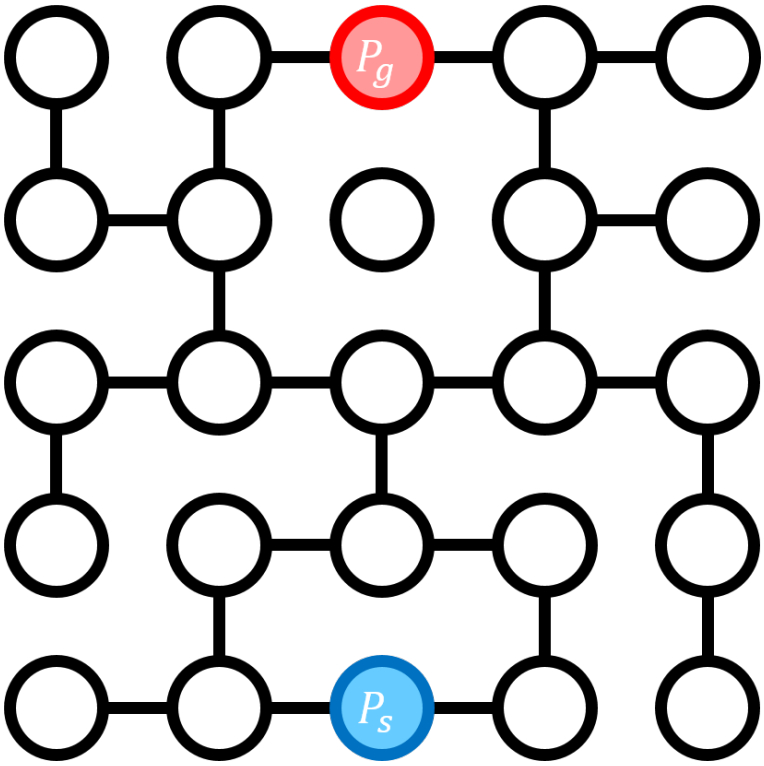

図 2.28　複数の次ノード情報と前ノード情報をともに含むグラフ一例

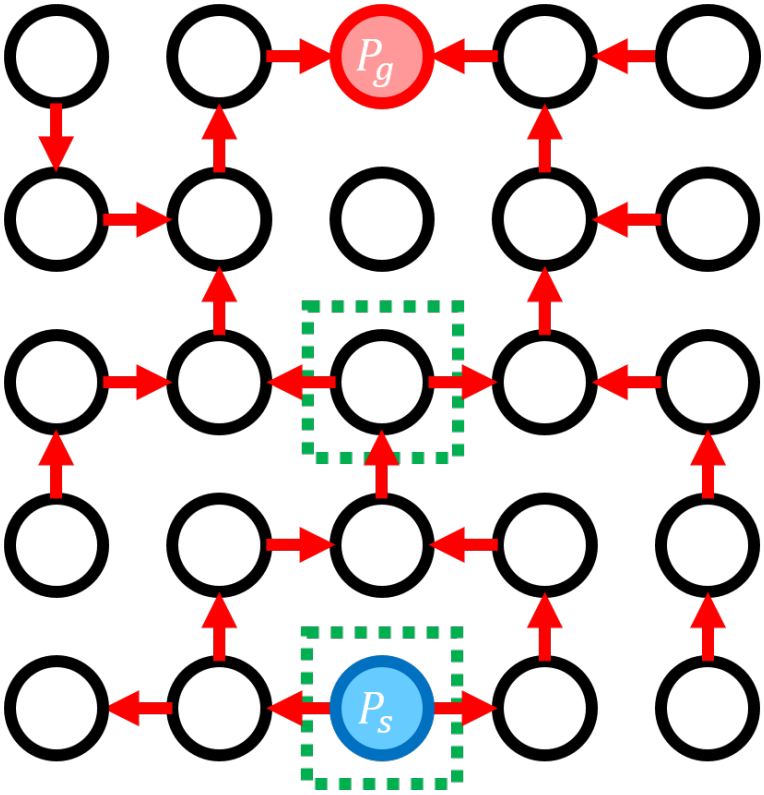

図 2.29　複数の次ノード情報を示したグラフ

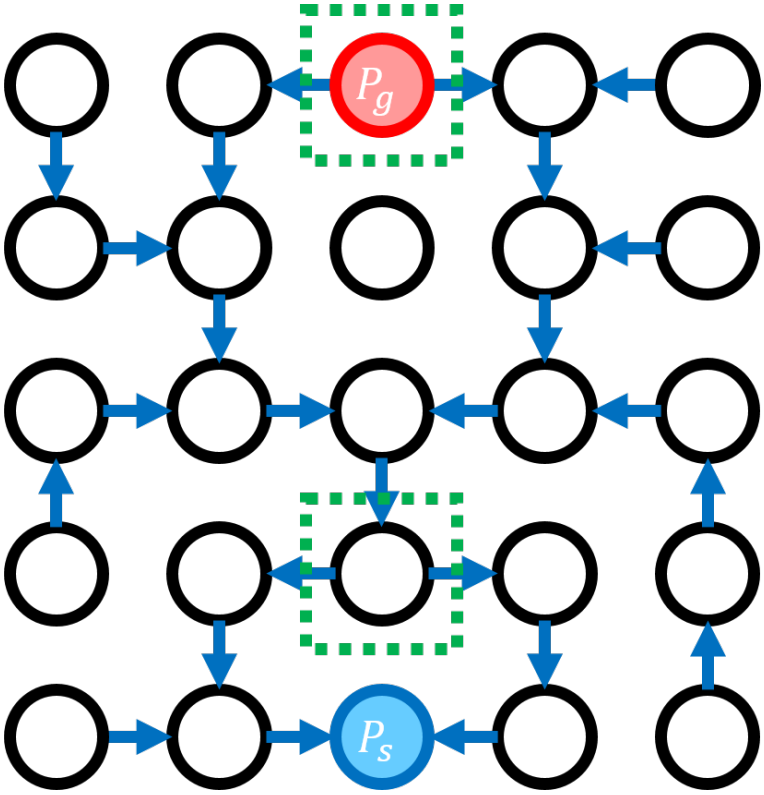

図 2.30　複数の前ノード情報を示したグラフ

2.2.6　2 回目以降の再探索処理

局所分析フェーズによって次ノード情報を局所的に再設定したが、このままでは不正確なコストが設定されているノードが存在する可能性がある。そのため、さらにマップ変更が行われた場合の局所分析フェーズに対応できない。スタート近接ノードから前ノード情報を参照する際に、本来再探索するべき処理を省いてしまう場合がある。そこで局所分析フェーズと全域分析フェーズの 2 つのフェーズをマルチスレッド処理により行う事で、次回以降の局所分析フェーズに頑健に対応することが出来る。

2.2.7　汎用グラフへの適応

冗長な処理を省く際の判定式 (2.6) を工夫することで、汎用的なグラフ構造に対応できる。スタートノードまでの距離とゴールノードまでの距離につり合いが取れるような新たな判定式にすることにより、汎用的なグラフ構造に対応した再探索が可能となった。図 2.31 は、汎用的なグラフ構造を示したものである。

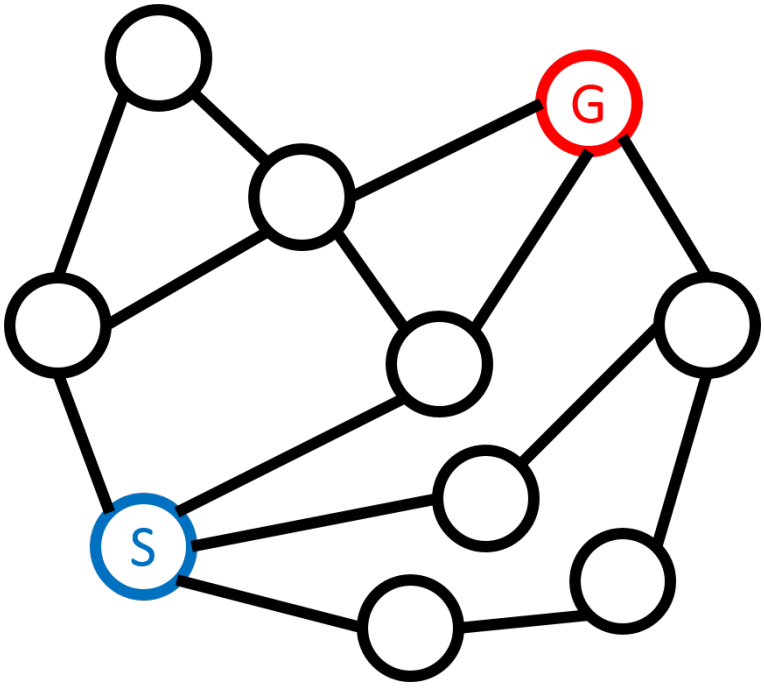

図 2.31　汎用的なグラフ構造の一例

第 3 章

実験と評価

3.1　本手法と従来のアルゴリズムの比較

本手法の有用性を評価するために、本手法と既存のアルゴリズムであるダイクストラ法と A*アルゴリズムで比較を行った。計測を行う場面として、新たな経路が出現するようなマップ変化時の最短経路再探索の処理時間を記録した。本手法における計測結果は、再探索のみにかかる時間と正確なコストマップの生成にかかる時間を記録した。計測するマップはパターンの違う 2 種類のマップで比較を行い、再探索処理を繰り返し行う事で複数の処理時間データを記録し、その複数の計測結果の平均値を取得した。また、経路検出フェーズと並行して行う全域分析フェーズを行わずに経路検出フェーズのみで繰り返し再探索を行う事が出来るかの検証も行った。

自由にマップの制御と経路探索が行えるプログラムを作成し、これを実験に使用した。新たな経路が出来るようなマップ操作を行い、再探索を実行する。各アルゴリズムで同様の操作を行った。

図 3.1 と図 3.2 は、実験をしたマップの 15 × 15 における例を示したものである。図 3.1 は、15 × 15 の蛇状のマップであり、図 3.2 は、15 × 15 の格子状のマップである。表 3.1 で、実験に用いた PC のスペックを示す。

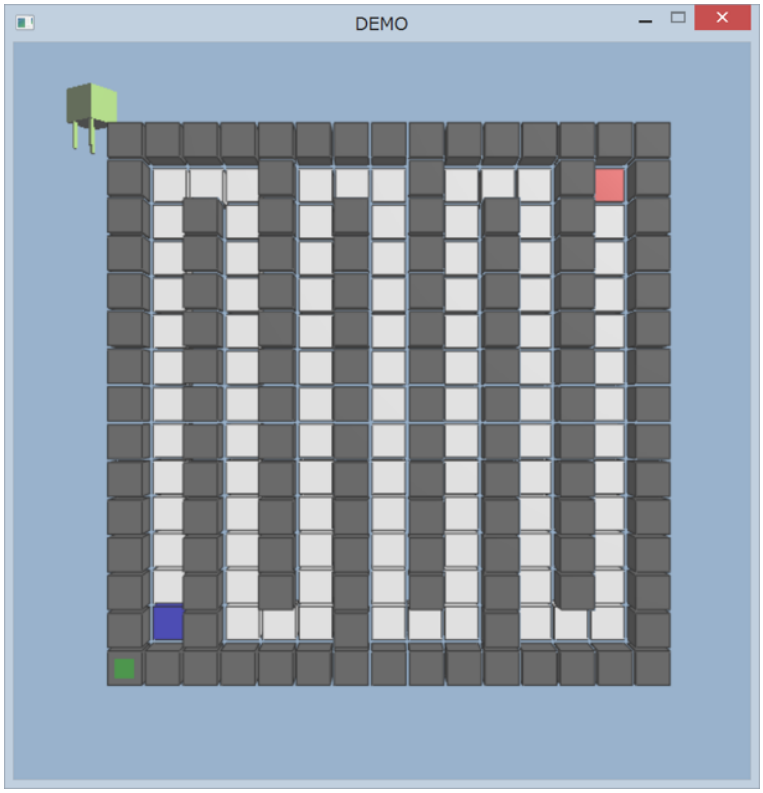

図 3.1　蛇状のマップ（15 × 15 の例）

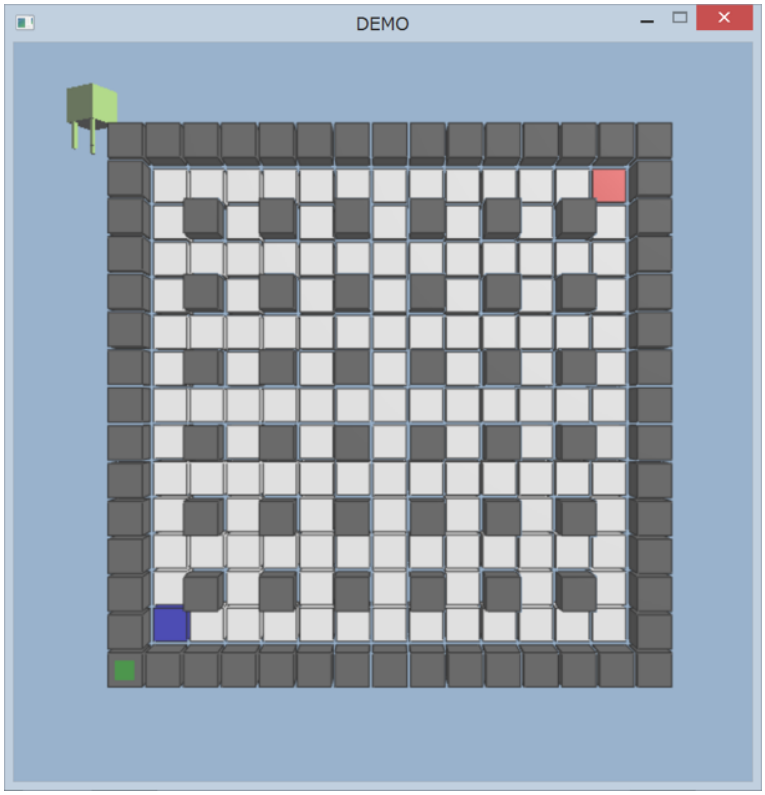

図 3.2　格子状のマップ（15 × 15 の例）

表 3.1　実験用 PC

CPU	Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
メインメモリ	16.0 GB
GPU	Intel(R) HD Graphics Family
解像度	1920 × 1080

3.2　実験結果

1003 × 1003 のマップで実験を行った。表 3.2 と表 3.3 で、それぞれのマップにおける結果を示す。表 3.2 は、蛇状のマップにおける実験結果を示しており、表 3.3 は、格子状のマップにおける実験結果を示している。また、2 つのマップにおいて、経路検出フェーズと並行して行う全域分析フェーズを行わずに経路検出フェーズのみで繰り返し再探索を行う事は、現状のアルゴリズム

では不可能であった。

表 3.2　蛇状のマップ実験結果

Dijkstra	A*	Our(正確なコストマップ)	Our(経路算出)
417.3	424.7	336	84.9

表 3.3　格子状のマップ実験結果

Dijkstra	A*	Our(正確なコストマップ)	Our(経路算出)
416.1	416.9	415.6	0

3.3　評価と分析

実験結果から局所分析と経路検出のみの処理時間は従来のアルゴリズムよりも速く処理できた。格子上のマップにおける処理時間が 0 であった理由として、新たな最短経路が現れないようなパターンであるため、冗長な再探索処理を省いたことでこのような結果になった。実験では再探索を繰り返し行ったが、処理時間は安定して同様の処理速度を示した。再探索にかかる処理時間は、ノードの接続情報を局所的に再設定することで高速に処理することが出来た。マップの形状によって本手法のパフォーマンスが変化することが分かった。

第 4 章

まとめ

本研究では、マップ変化時における再探索速度の高速化を目的とした。ゴールまでの距離と次ノード、スタートまでの距離と前ノードいう計 4 つの情報を各ノードに持たすことで、各フェーズでの効率的な処理を実現した。全域分析フェーズでは、4 つの情報を正確に設定し、局所分析フェーズでは、スタートまでの距離と前ノード情報を利用し次ノード情報を更新する。そして、経路検出フェーズでは、更新された次ノード情報を利用することで経路を得ることが出来る。

ダイクストラ法と A*アルゴリズムと本手法で特徴の違う 2 つのマップで比較をした。比較により、本手法の即時的な経路算出にかかる時間は、従来の手法よりも高速であることが分かった。正確なコストマップの算出を含めた処理時間では、従来の手法と同様の結果を示した。このことから、マップが変化した際の経路の再探索を高速に行うという点で非常に有効な手法であることが分かった。実験における格子上のマップの結果は新たな経路が現れない際の処理時間を示している。本手法では、再探索をする必要が無い場合に関しても判定式を用いることで、必要のない冗長な再探索を行わずに済むことが分かった。

本手法では、実験により、どのようなマップパターンであっても必ず全域分析フェーズを行う必要があることが分かった。このことから、正確なコストマップに対して局所分析フェーズを行う事が必須であるため、2 回目以降の再探索には必ず全域分析フェーズを行う必要がある。本手法では、マルチスレッド処理によりこれを実現したが、局所分析フェーズにより生成したコストマップを用いて再度、局所分析フェーズが行う事が出来れば、より目まぐるしいマップの変化に対応した経路探索が実現できたと考える。

また、他の経路探索手法では、スタートまたはゴールのどちらか一方が動いても、経路探索が可能である。だが、本手法では、ゴールからの距離とスタートからの距離の 2 つを利用しているため、経路探索中にスタートまたはゴールが動くような場合では、全域分析フェーズで得た情報を利用できない。そのため、スタート地点やゴール地点が動かない、あるいは、経路探索をするうえで影響の少ない軽微な移動をするような場面での利用が期待できる。

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] 株式会社長大. 交通シミュレーションによる効果の検証. https://www.chodai.co.jp/products/case/001987.html. 参照:2020.2.23.

[2] 東電設計株式会社. 津波避難に関する群集避難行動シミュレーション技術. http://www.tepsco.co.jp/introduction/disaster/simulation/index.html. 参照:2020.2.23.

[3] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, Vol. 1, pp. 269 – 271, 1959.

[4] 岩通ソフトシステム株式会社. ダイクストラ法アルゴリズム. https://www.iwass.co.jp/column/column-03.html. 参照:2020.2.18.

[5] Mojang. Minecraft 公式サイト. https://www.minecraft.net/ja-jp/. 参照:2020.2.18.

[6] KONAMI. スーパーボンバーマン r 公式サイト. https://www.konami.com/games/bomberman/r/jp/ja/. 参照:2020.2.18.

[7] D3PUBLISHER. 地球防衛軍 5. https://www.d3p.co.jp/edf5/. 参照:2020.2.18.

[8] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, Vol. 16, pp. 87 – 90, 1958.

[9] M. Desrochers and F. Soumis. A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR*, Vol. 26, pp. 191 – 212, 1998.

[10] S.E. Dreyfus. An appraisa l of some shortest path algorithm. *Operations Research*, Vol. 17, pp. 548 – 566, 1969.

[11] 大内斉, 五十嵐治一. 曲面評価関数を用いたサッカーエージェントの移動先決定. *The 21st Game Programming Workshop 2016*, 2016.

[12] I. Noda and H. Matsubara. Soccer server and researches on multi-agent systems. *Proc. of IROS-96 Workshop on RoboCup*, pp. 1 – 7, 1996.

[13] 秋山英久. サッカーシミュレーションリーグ第 2 回. 情報処理, Vol. 51, pp. 1303 – 1316,

2010.

[14] 横山秀, 金子知適. ゲーム木に基づく並列探索での下位曲面の分担. *The 21st Game Programming Workshop 2016*, 2016.

[15] K. Hoki and T. Kaneko. Large-scale optimization for evaluation functions with minimax search. *Journal of Artificial Intelligence Research*, Vol. 49, pp. 527 – 568, 2014.

[16] S. Yokoyama, T. Kaneko, and T. Tanaka. Parameter-free tree style pipeline in asynchronous parallel game-tree search. *14th International Conference on Advances in Computer Games, to be published in ICGA Journal*, 2015.

[17] 今川孝久, 金子知適. モンテカルロ木探索における子孫の勝敗確定時のプレイアウト結果の修正. *The 21st Game Programming Workshop 2016*, 2016.

[18] M. Winands et al. Monte-carlo tree search solver. *CG, Springer-Verlag,*, pp. 25 – 36, 2008.

[19] Ching-Nung Lin and Shi-Jim Yen. Accelerate deep learning inference with mcts in the game oh go on the intel xeon phi. *The 21st Game Programming Workshop 2016*, 2016.

[20] 吉村建志, 宝珍輝尚, 野宮浩揮. タブーサーチを用いた麻雀における最適行動の探索. *The 21st Game Programming Workshop 2016*, pp. 73 – 80, 2016.

[21] 水上直樹, 鶴岡慶雅. 期待最終順位に基づくコンピュータ麻雀プレイヤの構築. 第 20 回ゲームプログラミングワークショップ, pp. 179 – 186, 2015.

[22] 水上直樹, 中張遼太郎, 浦晃, 三輪誠, 鶴岡慶雅, 近山隆. 降りるべき局面の認識による 1 人麻雀プレイヤの 4 人麻雀への適用. 第 18 回ゲームプログラミングワークショップ, pp. 1 – 7, 2013.

[23] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimal cost paths. *IEEE Transactions on Systems Science and*

Cybernetics, Vol. 4, No. 2, pp. 100 – 107, 1968.

[24] Anthony Stentz. Optimal and efficient path planning for partilly known environments. *IEEE Internatinal Conference on Robotics and Automation*, 1994.

[25] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. pp. 156–165, 2005.

[26] Dorothea Wagner, Thomas Willhalm, and Christos Zaroliagis. Geometric containers for efficient shortest-path computation. *J. Exp. Algorithmics*, 2005.

[27] Emanuele Berrettini, Gianlorenzo D'Angelo, and Daniel Delling. Arc-flags in dynamic graphs. *9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09)*, 2009.

[28] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. *Internatinal Workshop on Experimental and Efficient*, pp. 319 – 333, 2008.

[29] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. *Transportation Science*, Vol. 51, No. 2, 2015.

[30] 石村園子. やさしく学べる離散数学. 共立出版株式会社, 2007.