(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5448827号
(P5448827)
(45)発行日　平成26年3月19日(2014.3.19)　(24)登録日　平成26年1月10日(2014.1.10)

(51)Int.Cl.　F I
G01C 21/34 (2006.01)　G01C 21/00 G

請求項の数 15　(全 44 頁)

(21)出願番号　特願2009-535133(P2009-535133)
(86)(22)出願日　平成19年9月18日(2007.9.18)
(65)公表番号　特表2010-508531(P2010-508531A)
(43)公表日　平成22年3月18日(2010.3.18)
(86)国際出願番号　PCT/GB2007/050558
(87)国際公開番号　WO2008/053240
(87)国際公開日　平成20年5月8日(2008.5.8)
審査請求日　平成22年9月10日(2010.9.10)
(31)優先権主張番号　0621508.1
(32)優先日　平成18年10月30日(2006.10.30)
(33)優先権主張国　英国(GB)

前置審査

(73)特許権者　507078083
コタレス・リミテッド
ＣＯＴＡＲＥＳ　ＬＩＭＩＴＥＤ
英国シービー２４・９ワイピー、ケンブリッジシャー、ケンブリッジ、ヒストン、ナロー・レイン６７番
(74)代理人　100084146
弁理士　山崎　宏
(74)代理人　100081422
弁理士　田中　光雄
(74)代理人　100118625
弁理士　大畠　康
(74)代理人　100176463
弁理士　磯江　悦子

最終頁に続く

(54)【発明の名称】ルートを生成するための方法および装置

(57)【特許請求の範囲】
【請求項１】
複数の点と複数のリンクとを有する加重有向グラフにおいて、ソースから目的地までの複数の多様なルートを装置によって生成する方法であって、
上記ソースから上記グラフの点の第１の集合へのソース・ルーティング木を生成するステップを備え、上記第１の集合は、少なくとも、上記ソースに隣接する点と上記目的地に隣接する点とを含み、上記ソース・ルーティング木は上記ソースから上記第１の集合の各点までのルートを有し、
上記グラフの点の第２の集合から上記目的地への目的地ルーティング木を生成するステップを備え、上記第２の集合は、少なくとも、上記ソースに隣接する点と上記目的地に隣接する点とを含み、上記目的地ルーティング木は上記第２の集合の各点から上記目的地までのルートを有し、
上記ソース・ルーティング木を生成するステップ及び上記目的地ルーティング木を生成するステップの後に、上記複数の多様なルートを形成するために上記ソース・ルーティング木と上記目的地ルーティング木とを合成するステップを備え、上記複数の多様なルートは、これらのルートの長さの所定の比率未満にわたって同じリンクを共有しており、
上記第１および第２の集合は同じ点を含み、
上記合成するステップで、上記ソース・ルーティング木および目的地ルーティング木に共通であり、かつ、上記ソース・ルーティング木および目的地ルーティング木が同じ方向に辿った複数の下位ルートを選択することを特徴とする方法。

【請求項２】
上記第１および第２の集合はそれぞれ同じ点からなる請求項１に記載の方法。
【請求項３】
上記ソース・ルーティング木は、上記ソース・ルーティング木において上記複数のルートから上記ソースの方向を指すバックポインタを含む一方、上記目的地ルーティング木は、上記目的地ルーティング木において上記複数のルートから上記目的地の方向を指すバックポインタを含み、
上記下位ルートは、上記ソース・ルーティング木および目的地ルーティング木の両方のバックポインタにより示されている隣接点のシーケンスであって上記ソース・ルーティング木および目的地ルーティング木のそれぞれで反対方向を指す隣接点のシーケンスを見つけることにより選択される請求項１に記載の方法。
【請求項４】
上記ソース・ルーティング木および目的地ルーティング木の点はコストと関連づけられ、上記下位ルートは、上記ソース・ルーティング木からのコストの総和と目的地ルーティング木からのコストの総和とが同じである隣接点のシーケンスを見つけることにより選択される請求項１に記載の方法。
【請求項５】
いかなる下位ルート上にもいかなる拡張下位ルート上にもない通過点を少なくとも１つ選択するステップと、
上記ソースから上記少なくとも一つの通過点を経て上記目的地までの最小限コスト・ルートを上記ソース・ルーティング木および目的地ルーティング木から算出するステップと、
上記１つの算出コストまたは各算出コストを、選択された下位ルートのうちの少なくとも１つのためのコストの同じ総額と比較するステップと
を備える請求項４に記載の方法。
【請求項６】
上記１つの算出コストまたは各算出コストと上記選択された下位ルートのための同じ総額との差を形成することを備える請求項５に記載の方法。
【請求項７】
算出コストと同じ総額との上記差が閾値より大きい上記１つの通過点または各通過点をはずすことを含む請求項６に記載の方法。
【請求項８】
各ルートに良好度を割り当てることを更に備える請求項１乃至７のいずれか１に記載の方法。
【請求項９】
上記良好度は、下位ルートの長さおよびルートの長さの関数であり、各下位ルートは、上記ソース・ルーティング木および目的地ルーティング木に共通であり、かつ、上記ソース・ルーティング木および目的地ルーティング木が同じ方向に辿る請求項８に記載の方法。
【請求項１０】
上記良好度に従って上記ルートのいくつかだけを選択することを更に備える請求項８または９に記載の方法。
【請求項１１】
上記グラフの各リンクは、該リンクを辿るコストと関連づけられており、上記コストは辿りのパラメータによって変化し、実行されるべき複数回の上記生成するステップのうちの１回目は、一般的な辿りパラメータ値のためのリンクのコストを算出して格納することを含み、これらの格納されたコストは、２回目の上記生成するステップ中に用いられる請求項１乃至１０のいずれか１に記載の方法。
【請求項１２】
上記グラフは道路網を表し、上記ルートは道ルートであり、

ユーザが点に接近するときに、複数のルートが分岐する当該点からのルートの選択に関する情報を提示することを備える請求項１乃至１１のいずれか１に記載の方法。
【請求項１３】
上記ルートのなかから選択されたルートに沿ってユーザにガイダンスを提供し、
ユーザが選択されたルートを離れるとき、この選択されたルートの下位ルートへのガイダンスをユーザに提供し、
各下位ルートは、上記ソース・ルーティング木および目的地ルーティング木に共通であり、かつ、上記ソース・ルーティング木および目的地ルーティング木が同じ方向に辿る請求項１２に記載の方法。
【請求項１４】
請求項１乃至１３のいずれか１に記載の方法を実行するためのコンピュータ・プログラム。
【請求項１５】
請求項１乃至１３のいずれか１に記載の方法を実行するようになっている装置。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、加重有向グラフにおいて複数の多様なルートを生成する方法および装置に関する。
【背景技術】
【０００２】
本明細書で用いられる用語「多様なルート（diverse routes）」とは、それらの長さの予め定められた比率よりも小さい比率で（通常はそれらの長さの８５％未満の比率で）同じリンクを共有するルートを意味する。
【０００３】
グラフＧ＝＜Ｖ，Ｅ＞は、頂点の集合（別名位置またはノード）ＶおよびエッジのEの集合（別名アークまたはリンク）Ｅから成る。エッジは、２つの頂点ｕおよびｖを接続する。ｖは、ｕに隣接していると言われている。有向グラフにおいては、各エッジは、ｕからｖまで方向性を有し、順序対（ordered pair）＜ｕ，ｖ＞またはｕ-＞ｖとして書かれている。無向グラフにおいては、エッジは、方向性を有しなくて、非順序対(unordered pair)｛ｕ、ｖ｝またはｕ＜－＞ｖとして書かれる。あらゆる無向エッジ｛ｕ、ｖ｝が２つの有向エッジ＜ｕ、ｖ＞および＜ｖ、ｕ＞により表される場合、無向グラフは有向グラフにより表されることができる。
【０００４】
有向および無向グラフには、重み付けをすることができる。重みは、各エッジに付与される。これは、例えば、２つの都市の間の距離、ドライブ時間、旅行の費用、電気径路（パス）の抵抗その他の、エッジと関連した量を表すために用いられることが可能である。特にグラフがある種のマップを表すとき、重みはエッジの長さまたはコストと呼ばれることがある。重みまたはパスまたはサイクルの長さは、重みまたはそのコンポーネント・エッジの長さの合計である。
【０００５】
この出願に記載されている方法は、通常、コストが加重有向グラフによって、記載でき、そのグラフに負のコストのサイクルがない如何なる領域にも適用可能である。我々は実施例の多くにおいて、ルーティングを道路網に用いている、しかし、この技術が他のドメイン、たとえば、交換点のネットワーク周辺でのパケットの中のルーティングや、集積回路、プリント回路基板または建物の配線のためのパスの発見等においても、等しくよく適用できることを理解すべきである。
【０００６】
道ルート・プランナは、ソースＡから目的地Ｂまでの最適ルートを見つけるように設計されている。それらは、これを最小限のコストを有する単一のルートとして定義する。こ

こで、コスト関数は、時間、ジャンクション遅延、距離、財政的なコスト、ルートを形成するリンクの道のタイプの単純な加重和（総和）である。これらの方法は、「最短パス」としばしば呼ばれているが、これは、コスト（それらが何を表すにせよ）の合計において、最短であることを意味し、必ずしもメートルにおいて、最短であることを意味するものではないと理解される。所与のコスト関数および道グラフに対して、ＡからＢへの最も低いコストが存在する。そして、これは周知のアルゴリズム、例えばダイクストラのアルゴリズムを用いることにより発見できる。退化ケースにおいては、グローバルに最低のコストを共有する複数のルートがある場合があるが、上記アルゴリズムはそれらのうちの１つだけを見つけ出す。ダイクストラのアルゴリズムは実際にＡからすべてのノードまでの最適ルートを捜し出す。しかし、そのアルゴリズムのバリエーションは、早期に終了しＢの方向を優先して調査するように設計されている。グラフへの主たる制約は、負の全体コストを有するサイクルがあってはならないことであり、それはすべてのコストが正である道路網について満足させるのに十分単純なものである。

【０００７】

異なるルート・プランナは、わずかに異なる道データベースおよびコスト関数を有する。それらは、ＡからＢまでの等しくもっともらしいルート（時々相互に非常に異なる）を見つけ出すことができる。すべてのこの種のもっともらしいルート（特に互いに非常に相違するルート）を見ることは、ユーザにとって便利である。

【０００８】

推奨されたルートがユーザが推測したか好んだものでないとわかることは一般にあることである。したがって、彼らは、時間および距離を比較できるように、ルート・プランナに彼らのルートを選ばせようとする。これは、道に沿って強制的なストップを加えることによって、または、ルーターが回避することになっている道リンク、ジャンクションまたは領域を指定することによって、または、コスト関数の加重を変えてより早い、より短い、より多くの高速道路ルートまたは他の基準の方を優先することによって、達成される。これは非常に時間がかかることがありえる。そして、しばしば、人は、他の人であれば試したであろうより良好なルートがあるのではないかと考えるままにされている。

【０００９】

例えば、ケンブリッジからマンチェスターまで移動することを考える。１つのルート・ファインダーは、Ａ１４、それからＭ６、Ｍ５６を用いることを提案する。別のものは、Ｍ６料金(M6 toll)を用いる。更に別のものは、Ａ１４それに続くＡ１を推奨する。これらの全ては、上手な人間のプランナが選ぶであろうルートの中にあるが、我々はそれ以外のものをどうやって見つけるのであろうか。

【００１０】

大部分の単一の最短路ルーターの基礎は、ダイクストラのアルゴリズムの何らかのバリエーションである。このアルゴリズムについての良いオリジナル参考文献は、E.W. Dijkstra. 「A note on two problems in connexion with graphs グラフに関しての２つの問題についての覚書」Numerische Matematik 1：２６９-２７１（１９５９）である。

【００１１】

これらのアルゴリズムは、ソースから任意のノードに着くためにそれまでに見つかった最も低いコストを維持する。それらは、（異なるヒューリスティック、例えばＡ＊アルゴリズムに基づいて）低コストを有する活動ノード（出リンクがまだ調査されることを必要とするもの）を繰り返し選択し、幾つかの出エッジを探査して、それらがこれまでに記録されたものよりも低いコストの別のノードに達することができるかどうかを見る。

【００１２】

このＡ＊アルゴリズムについての良いオリジナル参考文献は、P. E. Hart, N. J. Nilsson および B. Raphael 「A formal basis for heuristic determination of minimum path cost最小限のパス・コストの帰納的決定の形式的根拠」、IEEE Transactions on Systems Science and Cybernetics, 4：１００-１０７（１９６８）である。

【００１３】

今までに知られているのよりも安いコストでノードに達することができる場合、そのような各ノードは活動ノードのリストに加えられる。一旦アルゴリズムが終了すると、これらのアルゴリズムは、１グループ全体のノード（デスティネーションより低いコストを有するもの）への最も最低コスト・ルートを捜し出して、ソースから、目的地に着くことに役立つかもしれないすべてのノードまでのルーティング木（ＲＴ）を効果的に計算してしまっている。たとえどんな巧妙なプルーニングがこの木、例えばＡ＊（この木はソースおよび目的地に焦点を有する略楕円形状に終端する）に行われても、我々の方法はそれを多様な迂回ルート発見アルゴリズムに変換するために用いることができる。

【００１４】

ＥＰ１３３５３１５は、加重無向グラフにおける複数の非多様なパスを計画（プランニング）するための技術を開示する。ダイクストラのアルゴリズムは、二回用いられる。スタートからグラフのすべてのノードまでルーティング木を提供するために一回、そして、もう一回は、特にゴールからすべてのノードまでのルーティング木を提供するためである。そして、両方の木からの重みは合計され、グラフのあらゆるノードを介してスタートからゴールに行く重みを得る。位相的に異なるパスを見つけるべく、ルートは、障壁の存在下、位相的に相互へと変換でき得るかどうかについてテストされる。この種の技術はロボット・ガイダンスに特に適している。

【００１５】

ソースから目的地までＫ個の最短路を捜し出すように設計されたアルゴリズムの種類がある。実施例は、イェン（Ｙｅｎ）のアルゴリズム（制約のないＫ番目の最短パス）およびスアバレ（Ｓｕｕｒｂａｌｌｅ）のアルゴリズム（最短の素パス対）である（Ｊ．Ｙ．Yen．「Another algorithm for finding the K shortest loopless network paths　Ｋ個の最短無ループネットワークパスを見つけるための別のアルゴリズム」第４１回ミーティング会報において、アメリカ・オペレーションズ・リサーチ学会、vol．２０、１９７２、Ｊ．Ｗ．Suurballe, R.E．Tarjan．「A Quick Method for Finding Shortest Pairs of Disjoint Paths　最短の素パス対を見つける手っ取り早い方法」ネットワークス、１４：３２５３３６、ｐｐ　３２５-３３６、１９８４）。

【００１６】

イェンのアルゴリズムは良い代替ルートを見つけることに役立たない。なぜならば、Ｋ番目の最短パスは、最適ルートの軽微なバリエーションに過ぎないからである。それは、効果的にそれらのコストを無限に設定することによって、エッジおよびノードが禁止されるグラフ上でダイクストラのアルゴリズムを繰り返し実行することによって、働く。道路網において、これは真に多様なルートを見つけるにおいて、成功の見込みのないものである。というのは、禁止されたノードまたはリンクが良好な代替ルートのうちの１つに不可欠かどうか知る方法がないからである。

【００１７】

スアバレのアルゴリズムおよび他のものは、エッジ素パスまたはノード素パスを見つけるように設計されている。これらは良い代替ルートを見つけるためには役に立たない。なぜならば、幾つかの道路区分は、我々が見つけることを望む多様な代替ルートのいくつかに共有される場合があり、したがってそれらは素ではないからである。それらは、主要経路を見つけるためにダイクストラのアルゴリズムを走らせ、そして、リンク・コストの全てを変え、そして、予備ルートを見つけるために再びダイクストラのアルゴリズムを走らせることによって、機能する。

【００１８】

道ルートの発見のために、我々は、Ｋ個のルートのこの種の数学的に厳密な定義を必要としないかもしれない。

【００１９】

例えば、我々は、以前のルートにおいて、用いられた道リンクに対して加重することによって、連続するルートを見つけることができる。残念なことに、この種のルートは次第により装飾過剰になる。というのは、より多くのリンクが加重されており、多くの場合、

良いルートのいくつかは一般的な始めまたは終了を共有するが、それは、この技術によって、あまりに早く除外されるからである。

【００２０】

我々は、他のルートを生成することを目指して道リンク・コストまたはコスト関数を変えることができる。これは裏道路よりむしろ高速道路に対する好みのように、異なる利用者の嗜好を表すことに役立つことはありえる。しかし、それは、これらの特徴、例えばＭ６を用いるか、またはバーミンガムのまわりのＭ６料金を用いるかの選択、を共有する代替ルートを示さない。

【００２１】

システムの中には、ドライブ中、道リンクのサブセットのコストを変えて、ルートを再計算できるものがある。これはユーザが、迂回路を望むと言うためにボタンを押したか、または、いくつかのリンクを用いることのコストを変える新しい交通情報をシステムが受信したからかもしれない。これらの場合、現在のシステムは、単一のルーティング・アルゴリズムを別に走らせ、新規な最短パス（最低コスト）ルートにユーザを案内する。これは、通常、影響を受けたリンク周辺での短い迂回となり、第一のルートへ戻る結果となる。これは我々のものと非常に異なる技術である。なぜならば、それは、リンクのための新規なコストを用い、多様な選択肢ではなく単一の最も最適なルートを再び探すからである。

【００２２】

米国特許６１９９００９号は、異なる優先設定（最速、最短、組合せ）に基づいていくつかのルートを計算して、ユーザがそれらの間で選択を行うことを許容するための技術を開示する。このアプローチに関する問題は、ほとんどの場合において、ルートが相互に類似しているか同一でさえあって（例えば、最短ルートは、最速ルートであってもよい）、おそらく５％だけ長い根本的に異なるルートは示されないということである。

【００２３】

周知のシステムは、いかなるルート内（ｅｎ-ｒｏｕｔｅ）代替物を示すこともなく１本の選択されたルートに沿って、ドライバを導く。もちろん、ドライバが選択されたルートからそれる場合、多くの市販システムは新規なルートを再計算する。そして、米国特許５６７５４９２号は、いくつかの代替ルートを事前計算し、それらを表示する方法を示している。これらは、大抵いつも、以前の選択ルートへ戻る短い迂回路であり、完全に異なるルートのための選択とは解釈されない。

【発明の概要】

【００２４】

本発明の第１の側面によれば、加重有向グラフにおいて、ソースから目的地までの複数の多様なルートを装置によって生成する方法が提供される。この方法は、上記ソースから上記グラフの点の第１の集合へのソース・ルーティング木を生成するステップと、上記グラフの点の第２の集合から上記目的地への目的地ルーティング木を生成するステップと、上記ルートを形成するために上記ソース・ルーティング木と目的地ルーティング木とを組み合わせる、つまり合成する（combining)ステップとを備える。

【００２５】

上記第１の集合は、上記グラフの点の全てを含むことができる。

【００２６】

上記第２の集合は、上記グラフの点の全てを含むことができる。

【００２７】

上記第１および第２の集合は、ソースおよび目的地の両方に隣接するグラフの点を含むことができる。

【００２８】

上記第１および第２の集合は、同じ点を含むことができる。

【００２９】

上記グラフは、それぞれが使用の第１優先度を有するリンクの第１の集合およびそれぞ

れが上記第１優先度よりも高い第２の優先度を有するリンクの第２の集合を有することができる。上記点の第１の集合は、上記グラフにおける上記ソースを含む第１領域において、上記第１または第２の集合のリンクによって、相互接続されている点と、上記第１領域の外側にある上記グラフの第２領域において、上記第１の集合のリンクによって、相互接続されているが上記第２の集合のリンクによっては相互接続されていない点とを備えることができる。上記点の第２の集合は、上記グラフにおける上記目的地を含む第３領域において、上記第１または第２の集合のリンクによって、相互接続されている点と、上記第３領域の外側にある上記グラフの第４領域において、上記第２の集合のリンクによって、相互接続されているが上記第１の集合のリンクによっては相互接続されていない点とを備えることができる。

【００３０】

上記ソース・ルーティング木（以下、「ソース木」ともいう）および目的地ルーティング木（以下、「目的地木」ともいう）は、最小限コスト木であってもよい。

【００３１】

上記合成するステップは、上記ソース木および目的地木に共通であり、かつ、上記ソース木および目的地木が同じ方向に辿る各下位ルートを選択することを備えることができる。上記ソース木および目的地木はバックポインタを含むことができ、上記下位ルートは、上記ソース木および目的地木の両方のバックポインタにより示されている隣接点のシーケンスを見つけることにより選択されてもよい。あるいは、上記ソース木および目的地木の点はコストと関連づけられることができ、上記下位ルートは上記ソース木および目的地木からのコストの合計が同じである隣接点のシーケンスを見つけることにより選択されてもよい。上記合成するステップは、上記ルートのうちの１つを形成するために、必要に応じて上記ソース木および目的地木に沿って各下位ルートをソースおよび目的地まで延ばすことを更に備えることができる。

【００３２】

上記方法は、各ルートに良好度を割り当てることを更に備えることができる。上記良好度は、下位ルートの長さおよびルートの長さの関数でもよい。上記良好度は、下位ルートの長さおよびルートの長さの差の関数でもよい。

【００３３】

上記方法は、いかなる下位ルート上にもいかなる拡張下位ルート上にもない通過点（バイアポイント）を少なくとも１つ選択するステップと、上記ソースから上記少なくとも一つの通過点を経て上記目的地までの最小限コスト・ルートを上記ソース木および目的地木から算出するステップと、上記１つのまたは各算出コストを、選択された下位ルートのうちの少なくとも１つのためのコストの同じ総和と比較するステップとを含むことができる。上記方法は、上記１つの算出コストまたは各算出コストと上記選択された下位ルートのための同じ総和との差を形成することを含むことができる。上記方法は、算出コストと同じ総和との上記差が閾値より大きい上記１つの通過点または各通過点をはずすことを含むことができる。

【００３４】

上記方法は、上記良好度に従って上記ルートのいくつかだけを選択することを更に備えることができる。上記方法は、最も高い良好度のＮ個のルートを選択することを含むことができる。ここで、Ｎは、正の整数である。上記方法は、良好度が閾値より大きいルートの少なくともいくつかを選択することを備えることができる。

【００３５】

上記方法は、上記ルーティング木のうちの少なくとも１つを格納することを含むことができる。

【００３６】

上記グラフは、コストと関連したリンクを備えることができる。上記リンクのうちの少なくともいくつかの各々を辿るコストは、辿り(traversal)のパラメータによって、変化し得る。上記リンクのうちの少なくともいくつかの各々を辿るコストは、辿りの時間によ

って、変化し得る。実行されるべき複数回の上記生成するステップのうちの１回目は、一般的な辿りパラメータ値のためのリンクのコストを算出して、格納することを含むことができ、これらの格納されたコストは、２回目の上記生成するステップ中に用いられることが可能である。上記目的地ルーティング木を生成するステップは、上記ソース・ルーティング木を生成するステップの前に行われることができる。

【００３７】

上記ルートは、各ルートの少なくとも１つの特性に従って順序づけられていてもよい。上記特性は、良好度でもよい。

【００３８】

上記グラフは道路網を表すことができ、上記ルートは道ルートでもよい。

【００３９】

上記グラフは集積回路またはプリント回路を表すことができ、上記ルートは配線（インターコネクション）であってもよい。

【００４０】

上記グラフは配線設備を表すことができ、上記ルートはワイヤリング配線であってもよい。

【００４１】

上記方法は、選択されたルートおよび／または良好度の数に応じて上記配線の配置の順序を選択することを更に含んでもよい。選択されたルートの数が最も低いものについての配線が、最初に配置されてもよい。

【００４２】

上記グラフは通信ネットワークを表すことができ、上記ルートは通信パスである。上記通信ネットワークは、インターネットでもよい。

【００４３】

本発明の第２の側面によれば、本発明の第１の側面に係る方法によって、道ルートを生成することを含むナビゲーション方法が提供される。

【００４４】

上記方法は、ユーザに道ルートに関する情報を提示することを備えていてもよい。上記方法は、ユーザが点に接近するときに、複数のルートが分岐する当該点からのルートの選択に関する情報を提示することを備えてもよい。上記情報は、道路標識を表す形で表示されてもよい。

【００４５】

上記方法は、上記ルートのなかから選択されたルートに沿ってユーザにガイダンスを提供することを備えてもよい。上記方法は、ユーザが選択されたルートを離れるとき、この選択されたルートの下位ルートへのガイダンスを提供することを備えてもよい。

【００４６】

本発明の第３の側面によれば、本発明の上記第１または第２の側面に係る方法を実行するためのコンピュータ・プログラムが提供される。

【００４７】

本発明の第４の側面によれば、本発明の上記第３の側面に係るプログラムを格納しているコンピュータ可読媒体が提供される。

【００４８】

本発明の第５の側面によれば、本発明の第３の側面に係るプログラムの通信パスを横断する伝送が提供される。

【００４９】

本発明の第６の側面によれば、本発明の第３の側面に係るプログラムを実行するようにプログラムされたコンピュータが提供される。

【００５０】

本発明の第７の側面によれば、本発明の第３の側面に係るプログラムを格納したコンピュータが提供される。

【００５１】
本発明の第８の側面によれば、本発明の第１または第２の側面に係る方法を実行するようになっている装置が提供される。
【００５２】
このように、加重有向グラフにおいて、ソースから目的地への複数の多様なルートを生成する技術を提供することは可能である。それは、どれくらいのルートが必要とされるかとは無関係に、２つのルーティング木を生成することだけが必要である。
【００５３】
更に、それは、例えば、重み付け、あるいはコスト関数、あるいは個々のリンク・コストを変えることは必要でない。
【図面の簡単な説明】
【００５４】
【図１】図１は、小さい仮定的道路網を示す。
【図２】図２は、本発明の実施形態を構成している方法により生成されるソース・ルーティング木を示す、
【図３】図３は、上記方法により生成された目的地ルーティング木を示す。
【図４】図４は、図２および図３のルーティング木を合成した結果を示す。
【図５】図５は、図４から導き出された下位ルートつまり台地を示す。
【図６】図６は、図４から導き出されたルートの例を示す。
【図７】図７は、本発明の実施形態を構成している同じまたは類似の方法により生成されたソース・ルーティング木を示す。
【図８】図８は、上記方法により生成された目的地ルーティング木を示す。
【図９】図９は、図７および図８のルーティング木を合成した結果を示す。
【図１０】図１０は、上記方法により生成された多様なルートを良好性要因に応じたランキングとともに示す。
【図１１】図１１は、非常に単純なグラフのために同じまたは類似の方法によって、見つけられたソース・ルーティング木を示す。
【図１２】図１２は、同じグラフのための目的地木を示す。
【図１３】図１３は、図１１および１２のルーティング木を合成することにより生成された多様なルートを示す。
【図１４】図１４は、生成されたルートに関する情報をユーザに表示するための情報表示の例を示す。
【図１５】図１５は、ユーザに提示されるかもしれないナビゲーション・ディスプレイを示す。
【図１６】図１６は、道ジャンクションが上記方法の範囲内で如何にして分析されるかを示す。
【図１７】図１７は、本発明の実施形態を構成する装置のブロック略図である。
【図１８】図１８は、小さい仮定的道路網における優先道を示す。
【図１９】図１９は、本発明の実施形態を構成する修正された方法により生成されたソース・ルーティング木を示す。
【図２０】図２０は、上記修正された方法により生成された目的地ルーティング木を示す。
【図２１】図２１は、図１９および２０の木を合成することから導き出された下位ルートつまり台地を示す。
【図２２】図２２は、図２１の下位ルートから導き出された多様なルートを示す。
【発明を実施するための形態】
【００５５】
第一の例証および説明のために、多様なルート生成の単純な仮定的な例が初めに記載される。我々は、仮定道に沿ったジャンクション間の距離だけしか分かっていない表示を用いる。図１は、数十メートルでの注釈がその長さに付された道の小さいサブセットを示す

。
【００５６】
我々は、１つの道の端（ソースノード、「Source」と示される）から別の道の端（目的地ノード、「目的地」と示される）までのルートを計算し、最も短いルートだけでなく、良好な多様な代替物であるいくつかの他のルートも見つける。もちろん、この種の制限されたネットワークでは、代替物は残りの道を、多く取り上げる。しかし、何百万ものノードを有する本当のネットワークでは、我々は何億もの可能性の中から最良の僅かな代替物だけを見つける。
【００５７】
我々の次の仔細な実施の説明はより複雑な道路網を用いる。それは描画するのがより難しいもので、典型的な道は、２つのリンク（各方向において、１つ）により表される。我々は、それから、ターン遅延、各方向においての異なる状況および禁じられたターンを含む知識を組み込んで、各リンクから各考えられるサクセッサ・リンクへ行くコストをコード化する。
【００５８】
このように、我々は、我々の方法がノード・ベースまたはリンク・ベースのいずれの表示のためにも等しく良好に機能することを示す。それは、最短パス（最小限のコスト）木の計算に基づくいかなるルーティング・アルゴリズムをも強化するはずである。それは、どのようにしてそれらの木に到達したかということとは無関係であり、木を形成する最小限のコスト値およびバック・ポインタからのみ働く。たった２つの木の計算から、それは何千もの可能なルートを評価して、小さいサブセットを見つける。そのサブセットは、すべて良い代替物であり、各々局所的に最適であり、グローバルに多様であるサブセットである。
【００５９】
第一段階は、ソースノードから他の全てのノードまで最小限のコスト木を計算することである。これは、通常は、ダイクストラのアルゴリズムの異型またはＡ＊アルゴリズムを用いて実行されるが、このアルゴリズムは、計算の速度を上げたり格納要件を減らすために、しばしば幹線道路、予計算およびグラフ拘束条件を上手く使用することにより改良される。我々の例として、これの結果を図２として示す。
【００６０】
各道部分の各端において、我々は、ソースから最短パスを用いたその端までの距離を付記している。各道部分の各端から、たった１つの外向きの矢印が出ている。これは、最短パスを用いてソースの方へ戻る道を示す。我々は、これをバックポインタと言う。なお、ソース木については、それは、移動方向と反対の方向となる。これらは、ダイクストラのアルゴリズムまたはＡ＊アルゴリズムの必要な部分として計算されて、格納される。
【００６１】
例えば、ソースから目的地（距離３１０）として示されたノードへの最短パスは、距離が２８２、２４５、２０７、１９７、１３０、８５、２５、１４、そして最後に０であるノードを通って矢印をたどることにより、後方に追跡されることができる。これは、ソースから目的地への最短パスルートであって、ダイクストラのアルゴリズムまたはＡ＊アルゴリズムによれば、まさにこの方法でトレースされるであろう。
【００６２】
第二段階は、他の全てのノードから目的地ノードに最小限のコスト木を計算することである。これはちょうど以前のアルゴリズムの異型であり、その出力は図３として示される。今回は、最短パスに沿った目的地ノードまでの距離が付記されており、矢印（ここでも各道の端から一つだけ）は、最短パスを用いた目的地までの往きの道を示す。
【００６３】
例えば、距離２３４を有する左上のノードから目的地までの最短パスは、矢印をたどって距離が２２５、２１５、１７４、１１３、１０３、６５、２８、および０である隣接ノードを通ることにより見つけられる。この木も、グローバルに最も短いパスをコード化す

るが、これはソースノードから目的地ノードまでの矢印を追うことによって、見つけられ、常に、図２から見つけられるものと同一である。

【００６４】

第１の木（図２）が単一のノードから多くの他のものまでの最短パス・ルートをコード化する一方で、第２の木（図３）は多くの他のものから単一のノードまでの最短パスをコード化するという点で、両木間に微妙な差があることに注目されねばならない。

【００６５】

我々の方法における次のステップは、ノードごとに、各木からの最小限のコストを合計することである。我々は、これをコスト-総和(cost-sum)と呼ぶ。これの結果は、図４に示される。例えば、上部のナンバー４７１は、図２および図３の最上部の対応する番号２１８および２５３を加えることによって、得られた。

【００６６】

これらの数は、強力な解釈を有する。いかなるノードＰでも、それらは、ソースノードからノードＰを経た目的地ノードへの最短パス・ルートのコストである。このようにして、我々は、ソースから目的地までのグラフ中の他のあらゆるノードを経た最短パス・ルートの集合を計算してしまった。これはそれ自体において、強力な結果である。しかし、膨大な数のこの種のルートが存在する。そして、ソースから目的地までどのようにして到達するかを計画するとき、大抵それらは重要でない。

【００６７】

我々は、同じコスト－総和を有する隣接ノードのチェーンが幾つもあることに気がつく。もちろん、ソースから目的地まで最短パス・ルートに沿って位置するノードの全ては同じコスト－総和を有しなければならない。そして、それはまさに最短パス・ルートのコストである。しかしながら、別のこの種のチェーンがある。この例では、それらは、３１０、３３２、３３５および３９５のコスト－総和を有し、図５において、太い線により強調されている。あたかもコスト－総和が何らかの想像上の平面より上の高さを表すかのように、我々は同じコスト－総和を有する最大長のチェーンの各々を「台地」と呼ぶ。

【００６８】

これらの台地のチェーンを見つけるために、様々な公知のアルゴリズムが用いられることが可能であるであろう。我々は、典型例を説明する。

【００６９】

この例に関して、我々は、各ノードに、我々がそのノードを訪問したことを示す単一ビットを与えることから始める。なお、まず最初は、未訪問に対して０を与える。我々は、それから、あらゆるノードを次々に走査する。順番は重要でない。それが既訪問（１）として示されている場合、我々は走査において、次のノードへ移動する。ノードが未訪問（０）として示されている場合、我々がそれを訪問したことを示すためにそれを１に変える。このように新たにマークの記された各ノードに対して、それをノードＱと呼ぶとして、我々は、そのノードＱへの参照のみを加えることによって、当該チェーンにおける隣接ノードのリストを開始する。我々はそれから、ソース木におけるリンク（矢印）をたどって隣接ノードつまりノードＲ１まで進み、それが同じコスト－総和を有する場合には、我々はそれを既訪問としてマークすると共にリストにＲ１への参照を追加する。そしてＲ２などのために繰り返す。これが終わると、我々はノードＱに戻って、目的地木におけるリンク（矢印）をたどって隣接ノードＳ１に進む。Ｓ１でのコスト－総和がノードＱについて同じである場合、我々はリストにＳ１への参照を追加し、Ｓ１を既訪問としてマークする。我々は、それから目的地木におけるリンク（矢印）をたどってＳ１から新ノードＳ２まで進む。そして、今度も、それがノードＱと同じコスト－総和を有する場合には、それをリストに加えるとともに既訪問としてマークする。これが終わったき、我々は、ノードＱがその一部であるチェーンのノードの全てを備えるリストを有する。

【００７０】

この時点で、我々は、そのチェーンのための良好性要因を計算して、それが今までに見つけられた最良のｎ個のうちの１つである場合だけ、リストにそれを保存する。ここで、

ｎは、通常は５であるが、例えば１０００かもしれない。良好性要因については後で説明する。

【００７１】

それから、我々は、走査において、次のノードへと続く。

【００７２】

ソースおよび目的木が道路区分のチェーンを同じ向きに辿るときに、台地は形成される。これは、当該チェーンが、ソースから離れること及び目的地の方へ行くことの両方に役立つことを示す。これは、当該チェーンがソースから目的地まで行くことに役立つかもしれないことを強力に示している。この種のチェーンは、それらの近傍の最良の道を用いる傾向にあって、ソースから目的地まで行くのを助けるために整列配置される。何百万ものノードを有する現実の道路網には、この種のチェーンが何千もある。そして、それらの多くが非常に短い。

【００７３】

台地の完全なルートを作るために、我々は単に、上記ソース木および目的地木において、矢印をたどってソースノードおよび目的地ノード自体に戻りさえすればよい。図６は、合成コスト（それをＣＣと言う）が図５において、３９５であった台地のための関連の矢印を示す。このルートは、上記台地を含むソースから目的地への最短パス・ルートである。

【００７４】

その台地について、その長さ（それをＬと言う）は、単にソース木からのノードＡとノードＢでの値の差（図２から、２２１　－　１７０＝５１）又は目的地木からのノードＡとノードＢでの値の差（図３から、２２５　－　１７４＝５１）に過ぎない。

【００７５】

ソースから台地への最短パス・ルートは、ソース木からのＡおよびＢでの値のうちより小さいもの（図２から、min（１７０，２２１）＝１７０）に等しい長さ（それをＳＰと言う）を有する。したがって、ＳＰ＝１７０である。

【００７６】

台地から目的地への最短パス・ルートは、目的地木からのＡおよびＢでの値のうちより小さいもの（図３から、min（２２５，１７４）＝１７４）に等しい長さ（それをＰＤと言う）を有する。したがって、ＰＤ＝１７４である。

【００７７】

このように、台地を組み込む最適ルートの全長はＳＰ＋Ｌ＋ＰＤによって、与えられる。そして、それはこの場合１７０＋５１＋１７４＝３９５である。この値は正確にソースから台地のノードのいずれか一つを経た目的地への最短パス・ルートの長さでなければならない。そして、それを我々は合成木（図４）から合成コストＣＣとしてすでに見つけている。

【００７８】

ソースから目的地に着くために役立つ台地は、役立たないものより長い傾向にある。なぜならば、それは、その近傍の他のものと比較して、速くかつよく整列配置された長く延びたルートを示しているからである。したがって、我々は、Ｌのより大きい値を探している。

【００７９】

ソースから目的地まで着くために役立つ台地は、あまり長くないルートの一部である傾向がある。なぜならば、長い台地がソースおよび目的地の両方から長距離のところにある場合、我々は長い台地に興味がないからである。したがって、我々は、ＳＰ＋Ｌ＋ＰＤのより小さい値を探している。

【００８０】

したがって、台地の「良好性」の良い第１の評価は、台地の長さ引く台地が一部であるルートの長さである。そして、それはＬ　－　（ＳＰ＋Ｌ＋ＰＤ）＝　－　（ＳＰ＋ＰＤ）である。我々の例において、それは、－（１７０＋１７４）＝－３４４である。一般に、よ

り複雑な表示については、それは、台地を辿るコスト引く全てのルートのコストであろう。これは、負の数である。

【００８１】

長さ単位（または一般に、コスト単位）から独立してこの計算を行うために、我々は、それをグローバルに最も短いパス・ルートの長さ（コスト）で割る。それは、この場合３１０である。したがって、我々の原（未加工の）良好性（それをＲＧと言う）は－３４４／３１０＝－１．１１である。

【００８２】

これをより使い勝手よくするために、我々は、その順序を保存するいかなる関数によっても、それを変形できる。我々は、結果として生じる値を「良好性」または略してＧと言う。我々が以後用いる関数は、$G = 100 - 99^{-RG}$ である。

【００８３】

それは、最適ルートが良好性Ｇ＝９９を有するように選択される。台地の外側のルートが最適ルート長の最大８５％（ＲＧ＝－０．８５またはよりポジティブ）までであるルートはＧの値が５０より大きい。台地の外側のルートが最適ルートとほぼ同じであるルートは、Ｇの値が略１（これらは通常、数千もあり、それらは全て、グローバルに最適なルートの軽微なバリエーションである）である。そして、より悪いルートは１未満の値を有し、それは急速に負になり得る。

【００８４】

図５において、太く濃く示される他のチェーンは、以下の値を有する。

CC	SP	L	PD	-(SP+PD)	RG=-(SP+PD)/310	G
395	170	51	174	-344	-1.11	-64
335	99	136	100	-199	-0.64	81
332	56	78	198	-254	-0.82	57
310	0	310	0	0	0	99

【００８５】

ユーザに対する表示のために、我々は閾値としてＧの値を用いることができる。そして、Ｇ＞５０の値は良い結果を生じる。彼らがより多くのルートを見ることを望む場合、いくらか低い閾値、即ちＧ＞１０にして、常により多くのルートを保持できる。通常は、これは、より多くの代替物を求めるか、Ｇのための閾値を変えるユーザにより達成される。これは、一般ユーザには必要でない。

【００８６】

ケンブリッジからカンタベリーまでの完全なグレートブリテン道路網上のテストにおいて、Ｇ＞５０を有するルートは１つしかない。これは、ほとんど全ての長さについて高速道路（すなわち、Ｍ１１、Ｍ２５およびＭ２）を用いる。調査のために、我々は閾値を緩和した。しかし、Ｇ＜５０を有するルートは全く興味をそそらなかった。なぜならば、それらの高速道路は非常によく整列配置されており、西側はロンドンの貧弱な道により、そして東側は海に囲まれているからである。

【００８７】

用途によっては、Ｇ＞５０を有する多くのルートがあることがわかった場合、我々は、ユーザに負担を掛けすぎるのを回避するために、示された数を制限することを選択してもよい。通常はＧの値が最も高い５つのルートだけを示す。ここでも、ユーザは、彼らが望む場合には、このパラメータを変えることができるが、上位の５つが通常最も興味をそそるルートを包含していることがわかる。デスクトップ・ルート・プランナのためには、我々はＧ＞５０を有するすべてのルートを示すかもしれない。しかし、車両内またはＰＤＡで計画を立てているときには、我々は上位の５つだけを示すかもしれない。

【００８８】
グレートブリテン・ネットワークでのテストにおいて、これの例は、ケンブリッジからマンチェスターへのもので、Ｇ＞５０である良い代替ルートは８つあるが、トップ５は多分最も人気のある選択肢を含むだろう。
【００８９】
最小限のコスト・ルート（例えばダイクストラのアルゴリズムまたはＡ＊アルゴリズムにおいて、）を見つけるために用いられる関数は、単に移動距離よりもずっと複雑でありえる。ほとんどの状況において、それは、実際に、所用時間により支配されるが、例えば、道のタイプ、ドライバの精通度、財務費用、安全記録によっても、すでに知られている態様で、影響されもするかもしれない。我々は、台地の計算において、通常、最も利用可能な、時間から独立したコスト関数を用いる。それは単に、距離の代わりに適当なコストを道にラベルづけすることに過ぎず、しきい値処理を含むアルゴリズムの残りの部分は同じままである。
【００９０】
時間依存情報が台地の計算において、用いられることが可能である。しかし、１つのコストだけが各道部分と関連づけられることに留意する必要があるであろう。さもなければ、上記ソース木および目的地木は同じ部分に対して異なるコストを有するかもしれない。そして、それは我々が複合コスト（ＣＣ）値を用いてチェーンを確認(識別）するのを妨げることになるであろう。これを克服する方法が以下に示される。
【００９１】
時間依存情報を扱うための代替方式は、コストのうち時間非依存部分を用いて良好な代替ルートを計算し、次に、ユーザに提示するために時間依存要因を計算することである。時間依存要因は、渋滞による追加時間、可変料金コストおよび時間により変わる道路使用料を含む。このようにして、ユーザは、良い地理的ルートがどのように時間依存要因に影響を受けたかについて見ることができ、別の時間での旅の方が適当であるかどうかとか、どのルートを選択したいかなどを、自分自身で決定できる。例えば、ドライバの通常のルートは渋滞により最適ルートよりも５分遅くなったかもしれないが、彼らがその選択肢を与えられれば、彼らは彼らの通常のルートを選択することを望むかもしれない。別の日には、彼らは、別の方法で選択するかもしれない。コンピュータは、すべての状況の下で正しくこの決定をすることは決して十分にはできないであろう。したがって、ドライバは、コンピュータが各選択肢の距離、時間、コストその他を推定するという激務を行うことで、現実的な選択肢について情報を与えられるべきである。
【００９２】
コストが上記ソース木および目的地木との間で異なってもよい場合には（おそらくそれらが時間依存であり、辿り時間の見積もりは必然的に異なるため）、我々は台地を見つけるためにＣＣ値を用いることができないだろう。別の方法は、台地を形成するリンクを確認するために上記ソース木および目的地木（図２および図３における矢印）におけるバックポインタを用いることである。すなわち、上記ソース木および目的地木の両方において、同じ運転方向（しかし、矢印は反対方向）に辿られる連続的なリンクを見つけることである。ソース木において、一方向に、そして目的地木において、反対方向に矢印を有するいかなるリンクも、この種のチェーンの一部である。そして、台地は、上記と類似の走査テクニックを用いて、その特性を有するすべての隣接するリンクを合成することによって、見つけられる。
【００９３】
一旦我々が台地の数をＧ値を用いることにより減らして最も興味深いものに絞ってしまうと、我々は多くの他の方法でそれらをフィルターにかけることができる。例えば、我々は、利用者の好み（高速道路、ジャンクションはより少なく、より低料金、運転コスト、精通度）に基づいて、それらを順番に並べることを選択できる。我々は、どのように歴史的およびリアルタイムの交通情報が旅の所要時間についての我々の評価（見積）を変えたかをユーザを示すこともでき、それによっユーザは、ある特定のルートがなぜ今日最も速

いルートになったか、そして、それが、どのように以前用いられたかもしれない他のものに匹敵するかを理解できる。ユーザは、特にデスクトップで、マウスボタンをクリックすると、これらの要因のいずれかによって、良い代替ルートをソートする能力を与えられることが可能である。

【００９４】

我々は、通常は、臨時の使用用に２つの制御を上級ユーザに提示するだろう。１つは、Ｇが上回らなければならない閾値を設定するもの（デフォルト５０）、そしてもう１つは表示される代替物の数（デフォルト５）を制限するものである。

【００９５】

上記の我々の例において、上記の表は、Ｇ＞５０の閾値については、我々は３１０、３３２および３３５の合成コスト（ＣＣ）を有する台地から生成されるルートを推奨するだけであることを示す。なお、これらはたまたま、上記表における合成コストで最も低い値であるが、それは必ずしもそうである必要はない。また、すべてのこれらのルートは、グローバルに最適なルートの１０％以内にあるコストを有するので、ドライバだけに知られている他の要因は、関係する僅かな余分の距離を容易に上回るかもしれない、そして、彼らはそれらの全てを見たことを喜ぶだろう。

【００９６】

我々は通常、「良好性」の順にソートされたルートを提示する。そして、それは好都合なことに、第１のものが良好性９９を有するグローバルに最適ルートであることを意味する。我々は、他の計算された要因（例えば、表欄内の時間、距離、コスト）を提示することができ、ユーザが望む場合には、これら他の要因に従って彼らがソートし表示することを許容する。

【００９７】

ユーザの求めに応じて、次のリンク、または、計画ルートにおいて、ある距離だけ前方のリンク全てを禁止し、それらの新たな拘束条件を前提としてグローバルに最適なルートを１つ捜し出すことにより、代替ルートを計算できるシステムがすでにある。それらは、通常は問題域周辺で短い迂回ルートを見つけ、その後、本来のルートに戻る。但し、時々、それらは全く異なるルートを見つけるかもしれない。我々の技術はこの時点でも用いられることができ、より多くの選択肢を提示することもできる、しかし、この種の情報でドライバを混乱させることは賢明でないかもしれない。むしろ、我々の技術は、迂回機能（diversionary function）を補足する予備のオプションとして使われるようにすることができる。

【００９８】

更なる例証および説明のために、多様なルートの生成方法の比較的単純な実際の例を説明する。これは図７～１０において、示される。そして、これらの図は同じ縮尺および広がりで描かれたマップである。選択された例は、英国ケンブリッジから英国マンチェスターまでのルートである。

【００９９】

第一ステップは、ケンブリッジから他の全ての点まで最良のルート（最小限のコスト）を含んでいる木を計算することである。これは、図７において、示される。

【０１００】

木は印刷のために単純化されており、その主要枝だけを示している。木全体の葉を入れると、このスケールでは、ほぼ完全にその土地を埋め尽くしてしまうからである。この種の木を計算する方法は、当該技術において、周知であり、ダイクストラのアルゴリズムおよびＡ＊アルゴリズムを含む。

【０１０１】

第二ステップは、他の全ての点からマンチェスターまでの最良の（最小限のコスト）ルートを含んでいる木を計算することである。これは、図８において、示される。

【０１０２】

第三ステップは、両方の木において、同じ方向に、すなわち、図７において、ケンブリ

ッジから離れる方向へ、そして、図８において、マンチェスターの方へ、辿られる道を見つけることである。辿りの方向が重要であるので、そのような道は単に２本の木の重なりでない。結果として生じる道は、図９において、示される。

【０１０３】

第四ステップ（その出力が図１０において、示される）は、最も長い道のチェーンを図９から選択し、図７のソース木を用いてケンブリッジへ戻るまで、そして、図８の目的地木を用いてマンチェスターに戻るまで、各チェーンの終端点を連結することにより、それぞれのチェーンから完全なルートを生成することから始まる。ルートはそれからそれらの良好性（重なり合うチェーンのコスト引く全てのルートのコスト）に従って並べられる。そして、この場合、我々は上位５個（良好性の順に０番から４番まで）を表示することを選択した。

【０１０４】

本実施例における最終ステップは、ユーザに関係情報を表示することである。たとえば、表１に示されるような情報である。

表１

台地ID	記述	時間	距離
0	A14,A1,A57,M1,A628	２時間４０分	256km
1	A14,M6,A556,M56,A5103	２時間４１分	284km
2	A14,A1,A617,A623,A6	２時間５５分	249km
3	A14,M1,A50,A500,M6	２時間５０分	290km
4	A14,A1(M),A1,M62,A56	２時間５３分	297km

【０１０５】

次に、多様ルート発生器および車両ナビゲーション・システムにおけるそのアプリケーションのより包括的な例について説明する。見出しは、便宜のために提供される。

【０１０６】

＜ルーティング木＞

これらの役立つチェーンを見つけるために、我々はまず、Ａからすべてのノードまでの、そして、すべてのノードからＢまでのルートを見る。我々は、ダイクストラのアルゴリズムを用いてＡからすべてのノードまでの最適ルートを計算し、これらが単一の木構造つまり、Ａ（ＲＴａ）からのルーティング木に格納され得ることに気がつくことができる。その木では、各ノードまたはリンクは、最適ルートに沿って以前のものへのポインタを含む。図１１は、このような木を示すために単純な例を示す。ノードまたはリンクは、その点までの最適ルートのコストを含むこともできる。この木は、局所的に最適でありＡから離れるためによく整列配置される道を用いる傾向がある。「局所的に最適」であるとは、全体コストを低下させることができるルートを外れるマイナーな迂回路がないことを意味している。

【０１０７】

次に、我々は、すべてのノードからＢ（ＲＴｂ）までの最も最適なパスを格納する第２のルーティング木を計算する。この木は、局所的に速くかつＢの方へ行くのに役立つ道路区分に有利に働く。そして、図１２は図１１において、示される例のための、得られた木を示す。

【０１０８】

＜木の合成(combining)＞

今度は、我々は、両方の木において、同じ方向に用いられるノードのチェーンを見つけるために、２つのＲＴを合成する。これらのチェーンは、Ａから移動して離れるのにおいて、そして、Ｂの方へ移動するために役立つ。このようなチェーンから完全なルートを生成するために、我々は、ＲＴａを用いてチェーンの始めからＡへとたどって戻り、ＲＴｂ

を用いてチェーンの端からＢまで前進する。図１１の例のための結果生じたルートは、図１３に示される。このルートは、局所的に最適である。すなわち、ＡからＢまで行くのに全体コストがより低くなる小さいずれは存在しない。チェーンからのこのようなずれがある場合には、チェーンはより低コストのそれたルートで形成されていたのだろうし、ルートのたどられた端からこのような脇道がある場合には、適切なルーティング木（ＲＴａまたはＲＴｂ）が最適ではなかったのであろう。

【０１０９】

これらのチェーンから生成されるルートが局所的に最適であるにもかかわらず、それらがグローバルには最適でない点に注意する。一般に、グローバルに最適なルートは１つだけある。それは単一のルーター（例えばダイクストラのアルゴリズム）によって、見つけられるものである。２番目に最適なルートは、通常　１番目に最適なルートの軽微なバリエーションであって、コスト僅かに増加しているだけである。例えば、町で始まり町で終わる３時間の車の旅において、計算された旅行時間に僅か何分かを追加する、脇道による何百もの小さいずれが潜在的に存在する。それら全て並べ替えると、３時間１０分までの行程時間を有するルートを何百万も生成することになる。その後でようやく、我々は、町と町との間の非常に異なるルートを用いるものであって余分に１２分要する第１の旅を見るかもしれない。この旅にも潜在的に何百万もの小さいバリエーションがある。それらはすべて、ほんのわずかだけ高コストであるが、局所的に最適である。もちろん、それが１，０００，０００番目に短いルートである場合であっても、それは、依然として完全に現実的な代替物であるかもしれない。というのは、見積行程時間は、多分±１０分の誤差を有するからである。そして、我々が選択肢を提示される場合、我々は他の要因を考慮できる。例えば、我々がドライブするルートがどれくらい楽しいかとか、我々がそのルートにどれくらい慣れているかとか、この機会における財政的なコスト対時間の重要性とかである。

【０１１０】

もちろん、このようなチェーンは多くあるが、それらの数の上限はグラフ中のエッジの数である。しかし、それらは道路区分の重なり合うチェーン（それを我々は、台地と呼ぶ）の長さ（一般に、コスト）およびそのチェーンを用いる最適ルートのための道の長さ（コスト）によって、特徴付けられることが可能である。これのためのコスト情報はＲＴｓデータにおいて、すでに利用できるので、これは代替ルートの良好性を評価する非常に効率的な方法である。我々は、適正な良好性関数を発見した。この関数は、グローバルに最適なものから下方への最良の多様ルートを順序づけるだけでなく、どれくらいの代替物が本当に役立つかについて確認するために無次元閾値を提供する。

【０１１１】

この方法を用いることにより、我々は、ダイクストラのアルゴリズムを二回実行することにより支配される時間で、ＡからＢまでのＮ個の最良の多様な代替ルートを計算できる。我々は、ダイクストラのアルゴリズムを実行するよりも非常に少ない時間で、このようなルートを数千も評価し順序づけることができる。但し、通常は、良好性要因が閾値を上回るものは１ダース未満である。

【０１１２】

周知の単一のルーター（ＡからＢへの最も短い単一のパス）のいくつかの実施において、ダイクストラのアルゴリズムのバリエーションを用いることは、望ましいかもしれない。これらは、より良い状態のノードまたはエッジを探索するために、または、より早く終了するために、または主要道路だけを調べるために、または、同時にソースおよび目的地から外へ調査するために、特別なヒューリスティックを用いる。それらは、より速く動作したり、メモリの使用量を減らすために、これを行う。台地を見つけるためにＲＴを合成する我々の方法がＲＴｓがどのような方法で計算されたかということとは無関係であることを理解すべきである。したがって、これらのバリエーションが適切であれば、それらのより良い実行時間またはメモリ使用特性を利用しつつ、多様な代替ルートを産生するために、我々の方法をそれらに適用できる。

【０１１３】
＜コスト関数＞
最良の多様な代替ルート（選択ルートchoice route）を計算するために用いられるコスト関数は、周知の要因のいずれでも組み込むことができる。それは、時間および期間に対して敏感であってもよく、リアルタイムまたは歴史的な交通情報を組み込むことができ、利用者の好みを考慮することができ、財務情報（例えば道路使用料金）を用いることができる。
【０１１４】
しかしながら、好ましい実施例において、我々は、基本的なコスト関数を用いる。それは、殆ど時間に重みをおき、距離については余り参酌しない。これは、人がＡからＢまで行くために考慮するかもしれないルートの殆どを生成する。次に、我々は、ルートプランナにおいて、それらに各々のためのデータを提示することができる。そのデータは、時間、距離、財政的なコスト、渋滞情報およびその他の計算しうるあらゆるパラメータ（ターン数、安全性等）を与えるものである。我々はこれらを、１日の異なる時間、異なる曜日、異なるタイプの車両のすべてに対して非常に簡単に計算することができる。なぜならば、我々は、第一ステップにおいて、見つけた一握りの選択ルートだけを考慮しさえすればよいからである。いまや我々はユーザをループに入れたので、彼らはすばやく各ルートがどこに行くかを見ることができ、彼らが異なるコスト基準にどのような重みを与えることを望むかに基づいて、彼らを決定できる。
【０１１５】
我々は、道路利用者課金があたりまえになれば、これが特に重要になると思っている。なぜならば、すべての旅に対して、そしてもちろんすべてのユーザに対して正しい方法で時間、距離および費用負担のバランスをとることは可能でないからである。同じ日の同じユーザさえ、彼らがその旅を終えるための固定期限を有するかどうか、疲れてきているかどうか、または、ちょうど現金が不足したかどうかによって、優先するものが異なるかもしれない。彼らは、かなり短い非高速道路ルートを通常好むかもしれないが、代替物が多くのターンを伴う場合、彼らは気分を変えるかもしれない。
【０１１６】
＜選択肢提示＞
周知のルート・プランナに関する他の問題は、ユーザがいくつかの他のルートがなぜ推奨されなかったかについて疑問に思うかもしれないということである。彼らに代替ルートをそれらの特徴とともに示すことによって、ユーザがそれらの代替ルートがどのように最良のものに匹敵するかについて見て、適正な選択をすることは、容易である。例えば、いつものルートはいつになくひどい渋滞になるかもしれない、あるいは、バイパスの建設のため代替ルートが好ましくなったかもしれない。
【０１１７】
ルート選定(route planning)ツールにおいて、我々は、異なる色を用いて異なるルートを強調することができ、各ルートについて時間、コストなどの関連した詳細を有するテキストボックスを提示することができる。先に説明されたケンブリッジからマンチェスターまでのルートに対するテキストボックスの例が図１４に示される。マップ上のルートを選択すればテキストボックス内の関連した線を強調することができ、テキストボックスにおいて、ある線を選択すればマップ上の関連したルートを強調することができるであろう。このようにして、ユーザは、我々が与えることのできるできるだけ多くの情報でもってルートを選択できる。
【０１１８】
ルート選定ツールにおいて同時に選択ルートを提示する代わりに、我々は、他のユーザーインターフェースを考慮できる。例えば、ルート内（en-route）ガイダンスとしては、我々は、代替物を示すマップでドライバを混乱させたくない。むしろ、行程内で選択が行われることができる箇所（それを、我々は選択点と呼ぶ）を計算し、彼らがその選択点に接近するときにドライバに知らせる。ケンブリッジからマンチェスターへの６本の最善ル

ートの例に対しては、ドライバは、彼らの旅の間、多くても３回選択肢が提示される。更に、これらは、利用可能な情報の関連サブセットとともに提示されることができる。それらは、おそらくジャンクション図表として、どちらかといえば図１５に示したような沿道のジャンクション看板のように、提示されることができる。この種の図表は、自動車のナビゲーション・システムにおいて、よく使用されており、最適ルート上にとどまるためにとるべき正しいターンをドライバに示す。選択点に接近するとき、我々のシステムは異なる。我々は、最適ルートと強調された他のあらゆる選択ルートの両方を有するジャンクション図表を示す。我々は、これを、用いられた道の概要、彼らの目的地までの相対時間、財政的なコスト等の情報で強化する。これらは、道路標識の形で示されることができ（そして、これ以上気をそらせるものであってはならない）、音声合成を用いて読出しすることもできる。それは「ロータリーで、Ｍ６およびＭ５６用の第１の出口を出てください。あるいは、１０分間長いけれども７ポンド安くするために、Ｍ１およびＡ６プライム２用の第２の出口を出てください”といった形をとることができる。交通情報が考慮された場合であってユーザが最適ルートがなぜ彼らが予想したものでないかについて疑問に思っているかもしれない場合に、特に役立つであろう。保存された時間またはコストは、これを彼らに明らかにする。

【０１１９】

選択ルート(Choice Routes) が一旦わかれば、ソース・リンクから目的地リンクまで少数の選択ルートをたどることによって、選択点(Choice Points)自体を発見するのは容易である。我々はソース・リンクから始める。そして、それはあらゆる選択ルートの初まりで発見される。あらゆる選択ルート上の次のリンクが同じである場合、我々はそれへ移動する。次のリンクがルートで異なる場合、我々は選択点に達しているので、あたかも我々がちょうど今再びソース・リンクから始めたかのように、各ルートを別個にたどる。各ルートが目的地リンクに達したとき、我々は終了する。我々は、旅を開始するときに選択点の全てを計算できる、または、常に少なくとも１の選択点を前もって知るように保ちつつ、行く道中で、それらを計算できる。

【０１２０】

選択点を計算するために用いられることが可能である他の方法がある。例えば１つの選択ルートにより用いられたリンクの全てをマークしする。そして、どこで選択ルートが、マークされたノードからマークされていないノード（選択点）へ変更され、そして、また、マークされていないノードへ変更（収束点）されるのかに注目して、他の各選択ルートをマークする。これらの方法のいずれも選択点を決定するために用いられることができ、アプリケーションの正確な必要性に従って選択されることができる。

【０１２１】

＜建設的、非破壊的＞

我々の方法の重要な特性は、それが建設的であり、同時にすべての良い道を見つけ、非常に良い重みを用いて、しかも、個々の道リンクのコスト要素を盲目的に変える必要なく、それらの道を評価するということである。重みまたは個々のコスト要素を変えて最も最適なルートをあまり最適ではなくさせる方法は、その他の未知の良いルートの最適性を低めなかったかどうかわからない。したがって、それは決して見つからない。

【０１２２】

我々は、この種の方法で我々の方法が第１に発見されるべきものであると思っている。なぜならば、我々の調査において、見つけた他のすべてのものが重みまたは個々の道要素を用いるコストの変更を伴うからである。以前に見つかったルートで用いられた一つ以上の道要素を禁止すること（人工的にそれらに無限のコストを与えることに等しい）によって、動作するものさえある。

【０１２３】

＜道グラフの表示＞

重み付けをされた有向グラフのために用いられることが可能である様々な表示がある。例えば道路網のために、我々は、ジャンクションをグラフにおけるノードとして、ジャン

クション間の道リンクをグラフにおけるエッジとして表すかもしれない。道を辿るコスト（例えばジャンクション間の道のりを辿るためにかかる距離および平均時間）を表すために、我々は、エッジにコストを割り当てることができる。あるいは、ジャンクションを通過するコスト（例えば、まっすぐに進行するためにかかる時間または左折若しくは右折するためにかかる時間）を表すために、我々はノードに対して、そのノードに着くために用いられるエッジ及びそのノードを離れるために用いられるエッジに依存するコストを割り当てることができる。この表示の多くのバリエーションは可能である。そして、我々の方法はその全部に適用できなければならない。

【０１２４】

この実施のために、我々は、ルーティング木を計算するために特に高速である表示を用いる。この表示は、ノードとして道ジャンクションを表さない。その代わりに、各エッジは片方向である。したがって、二方向の道（往復道）の伸びは２つのエッジ（各移動方向につき１つ）により表される。そして、これらは厳格にグラフのノードを形成する。これにより我々が移動方向ごとに異なる遅延をコード化することができる。そして、それは渋滞の時間に重要かもしれない。我々は、各エッジを「リンク」と言う。

【０１２５】

ジャンクションの特性をコード化するために、各リンクは、グラフにおいて、それから達することができる次のリンクへのポインタの組（集合）を有する。これらは、「次リンク」と呼ばれている。これらの次リンク・ポインタは、厳密にグラフのエッジである。第一のリンクから次のものへ移動するコスト（時間、距離、財政的なコストその他において、）は、このような各ポインタと関連づけられる。我々の実施は、１つのリンク・エッジの中間点から別のものの中間点へ移動するのにかかられる時間（次リンク時間）を用いることを選択する、しかし、他のものは、リンクの起点またはそれら終点を用いることを選択するかもしれない。結果に差はない。我々は、各リンクの長さ（「リンク距離」）を格納し、長さを加えて２で割ることによって、１つのリンクの中央から次のリンクの中央までの距離を計算する。このようにして、我々は、１つの「次リンク」において、ある長さの道を辿るコストおよびある特定の方向転換のためのジャンクションを用いるコストの両方をコード化した。それにより、あるソース・エッジまたはノードから他のエッジまたはノードまでのルートの全てを計算することが特に効率的なものとなる。

【０１２６】

ソース・リンクからの最適ルートよりもむしろ特定の目的地リンクへの最適ルートを計算するために、我々は各リンクにＰｒｅｖＬｉｎｋｓの集合を与える。それは、そのリンクへすべての可能な先行リンク（すなわちそこへ入れる道の部分）から入るコストをコード化する。あらゆるＰｒｅｖｌｉｎｋは対応するＮｅｘｔＬｉｎｋを有するので、幾らかの冗長がある。したがって、我々はいくつかの表示よりも多くのメモリを用いるが、実行時間は少ない。

【０１２７】

図１６は、脇道がジャンクションの方へ一方通行である道ジャンクションを示す。Aと記された道部分は２つのリンク１および３により表される。それは対面（２方向）道路だからである。同様に、道部分Ｂは、２つのリンク２および４により表される。道部分Ｃは、ジャンクションへ向かう一方通行であるので、１つのリンク５だけで表される。

【０１２８】

さて、道部分Ａ上のジャンクションの方へ移動している車両は、部分Ｂだけに続くことができる。そのため、１つの「次リンク」（矢印）だけがリンク１から出ており、そのリンクはリンク２を指している。同様に、道部分Ｂ上のジャンクションの方へ移動している車両は、部分Ａに続けることができるだけである。そのため、リンク４から次リンクが出ており、それはリンク３を指している。ジャンクションの方へ道部分Ｃを移動している車両は、部分Ａまたは部分Ｂのどちらかに折れてジャンクションから離れることができるので、対応するリンク５はそれから出ている次リンクを２つ有する。それらはリンク２，３を指している。ここでの重要な要因は、ＮｅｘｔＬｉｎｋｓが、それらが向かうリンクか

らではなく、それらが出ているリンクから、列挙されることができるということである。
【０１２９】
ＰｒｅｖＬｉｎｋｓは、次リンクの逆であって、ここでも、それらが出ているリンクから列挙される。それらは、車両がそこからこのリンクに入るかもしれないリンクを示す。例えば、車両は、このジャンクションで道部分ＢまたはＣから道部分Ａに入るかもしれないので、リンク３は、リンク４およびリンク５を指すＰｒｅｖＬｉｎｋｓを有する。
【０１３０】
このリンク・データは一組のファイルすなわちファイルの集合（各ファイルにはタイプ別の情報）として格納される。それらは、効率的にアクセスできるようメモリに簡単にマップされることができる。各リンクは単純な整数ＩＤ（長さ３２ビット、０から順に大きい数字）を与えられる。そしてそれは、さまざまなファイルへのオフセットとして用いられる。我々は、例としてリンクＩＤ４３５を用いる。リンク４３５の長さは、ファイルＬｉｎｋＤｉｓｔａｎｃｅの４３第五の１６ビット語で見つかる。それは、１６ビットのメートル単位の長さとしてコード化されている。ＮｅｘｔＬｉｎｋｓであるリンク４３５から達することができるリンクは、ＮｅｘｔＬｉｎｋｓと呼ばれるファイルに３２ビットのリンクＩＤを有する。ファイルＮｅｘｔＬｉｎｋＩｎｄｅｘにおいて、４３第五の３２ビット語はファイルＮｅｘｔＬｉｎｋｓにオフセットを与える。そのファイルには次リンクのｌｉｎｋＩＤｓが隣接して見つかる。それらの数は、ファイルＮｅｘｔＬｉｎｋＣｏｕｎｔにおいて、４３第五の８ビット語で見つかる。特定の次リンクへ移動するためにかかる時間は、ファイルＮｅｘｔＬｉｎｋＴｉｍｅにおいて、与えられる。ＮｅｘｔＬｉｎｋＩｎｄｅｘおよびＮｅｘｔＬｉｎｋＣｏｕｎｔの見出しも付けられている。さらに、我々は、リンクの数（ｎＬｉｎｋｓ）、次リンクの数（ｎＮｅｘｔＬｉｎｋｓ）およびＰｒｅｖＬｉｎｋｓの数（ｎＰｒｅｖＬｉｎｋｓ）を格納するマスター・ファイルを加える。一般に、ｎＰｒｅｖＬｉｎｋｓ＝ｎＮｅｘｔＬｉｎｋｓである。
【０１３１】
これまでのファイル定義を示す。ファイル内の要素の数は、ファイル名の後、角括弧内に示され、その後には、各要素のタイプが続いている。
【０１３２】
ファイル記述：
Ｍａｓｔｅｒ［３］：ＩＮＴ３２：リンクの数（ｎＬｉｎｋｓ）と、次リンクの数（ｎＮｅｘｔＬｉｎｋｓ）と、前リンクの数（ｎＰｒｅｖＬｉｎｋｓ）とを格納し、そして、おそらく、ファイルがどのようにして生成されたかについてのその他のトップ・レベル情報も格納している。これまでのところ、全３フィールドは、ＩＮＴ３２である。
【０１３３】
すべての他のファイルは、単純アレーである。リンクは、そのリンク・インデックス、すなわち０とｎＬｉｎｋｓ－１との間の整数により定義される。それは、長さが［ｎＬｉｎｋｓ］の倍数であるファイルに格納されたアレーの全てを指し示している。
ＮｅｘｔＬｉｎｋＩｎｄｅｘ［ｎＬｉｎｋｓ］：ＩＮＴ３２：次リンクが見つけられるNextLinksファイルにおけるオフセットを格納する。
ＮｅｘｔＬｉｎｋＣｏｕｎｔ［ｎＬｉｎｋｓ］：ＩＮＴ８：次リンクの数を格納する。
ＮｅｘｔＬｉｎｋｓ［ｎＮｅｘｔＬｉｎｋｓ］：ＩＮＴ３２：１つのリンクからアクセスされることができる次リンク（インデックスによって、）を格納する。
ＰｒｅｖＬｉｎｋと呼ばれている同様の１組の３つのファイルは、目的地ルーティングのためのルーティング木をコード化する。
ＰｒｅｖＬｉｎｋＩｎｄｅｘ［ｎＬｉｎｋｓ］：ＩＮＴ３２：次リンクがあるNextLinksファイルにおけるオフセットを格納する。
ＰｒｅｖＬｉｎｋＣｏｕｎｔ［ｎＬｉｎｋｓ］：ＩＮＴ８：リンクの数を格納する。
ＰｒｅｖＬｉｎｋｓ［ｎＰｒｅｖＬｉｎｋｓ］：ＩＮＴ３２：１つのリンクからアクセスされることができるリンク（インデックスによって、）を格納する
ＬｉｎｋＴｙｐｅ［ｎＬｉｎｋｓ］：ＩＮＴ８：道リンクのタイプ。０は速い、２５５は

遅い。
ＮｅｘｔｌｉｎｋＴｉｍｅ［ｎＮｅｘｔｌｉｎｋｓ］：ＩＮＴ３２：このリンクの中央から次リンクの中央まで行くのにかかる増加時間（ミリ秒）（ジャンクション遅延を含む）。
ＰｒｅｖＬｉｎｋＴｉｍｅ［ｎＰｒｅｖｌｉｎｋｓ］：ＩＮＴ３２：前のリンクの中央からこのリンクの中央まで来るのにかかる増加時間（ミリ秒）（ジャンクション遅延を含む）。
ＬｉｎｋＤｉｓｔａｎｃｅ［ｎＬｉｎｋｓ］：ＩＮＴ１６：このリンクを用いたときの増加距離（メートル）。
【０１３４】
＜ルーティング、アルゴリズム、ソース木＞
所定のリンクから他の全てのリンクまでの最小コスト・ルート（ソース・ルーティング木）を造るために用いるルーティング・アルゴリズム（ダイクストラのアルゴリズムの一バージョン）は、以下の通りである。
【０１３５】
そのリンクに達するためにこれまでに見つかった最小限のコストであるｍｉｎ－ｃｏｓｔを有するすべてのリンクから始める。そのリンクの原点（ｍｉｎ－ｃｏｓｔがゼロである）以外のすべてのリンクについてそれを無限とする。活動ノード（我々がそれらの出リンクの全てをまだ探索したわけではないもの）の可変長リストを造る。最初にこのリストにソース・リンクだけを入れる。
【０１３６】
これを繰り返す。
アクティブリストが空であれば、我々は終了しており、リターンする。
【０１３７】
そうでなければ、アクティブリンクのリストから第１のアクティブリンクを削り、それをｌｉｎｋＰと呼ぶ。それは、我々がそれに着くことができることを知っている最小コストｃｏｓｔＰを有する。出リンクｌｉｎｋＰＮごとに、ｌｉｎｋＰからｌｉｎｋＮへ行く増分費用ｃｏｓｔＰＮを加える。そして、それがその出リンクに関して格納された最小コスト（ｃｏｓｔＮ）未満かどうか確かめる。もしもそうであるならば、ｃｏｓｔＮを低い値で更新し、ｌｉｎｋＰを参照するためにｌｉｎｋＮのバックポインタを更新し、ｌｉｎｋＮをアクティブリンクのリストに加える（それがまだそこにない場合）。
【０１３８】
このアルゴリズムが終わると、我々はルーティング木を構築したことになる。
【０１３９】
＜ルーティング木の格納＞
我々の実施例において、我々は前に計算されたルーティング木を再利用したいかもしれないので、我々は３つのファイルとしてそれぞれをファイリングシステムに格納する。異なる木は、各ファイル上の後置整数値＜ｎ＞を用いることにより区別される。例えば、ＴＴＢａｃｋＩｎｄｅｘ１には、木１のためのバックポインタが入っている。規約により、現在選択されているソースおよび目的地リンクのための木は、ｎが０および１であるファイルに格納される。ここでも、ファイルの長さは、ファイル名の後、角括弧内に示されている。
【０１４０】
ファイル記述：
ＴＴＭａｓｔｅｒ＜ｎ＞［４］：一連のＩＮＴ３２：ｎＬｉｎｋｓ、ｆｒｏｍＳｏｕｒｃｅ（０＝目的地、１＝ソース）ｒｏｏｔＬｉｎｋＩｎｄｅｘ、ｒｏｏｔＳｕｂｎｏｄｅＩｎｄｅｘ。
ＴＴＢａｃｋＩｎｄｅｘ＜ｎ＞［ｎＬｉｎｋｓ］：ＩＮＴ３２：木＜ｎ＞について、最も低いコストでこのリンクに到達するために用いられたリンク。次または前の（ソースまたは目的地）木のための作業。

ＴＴＭｉｎＣｏｓｔ＜ｎ＞［ｎＬｉｎｋｓ］：ＩＮＴ３２：木＜ｎ＞について、このリンクに到達するために見つかった最小限のコスト。

【０１４１】

＜最適ルートをたどる（トレースバックする）こと＞

ソース・リンクｌｉｎｋＡから目的地リンクｌｉｎｋＢまでの最適ルートを見つけるために、我々は、ＴＴＢａｃｋＩｎｄｅｘ０ファイルを用いなければならないだけである。例えば、ｌｉｎｋＢがＩＤ６７８，１２３を持っていたと仮定する。最適ルートに沿って我々をｌｉｎｋＢに導いたリンクは、ファイルＴＴＢａｃｋＩｎｄｅｘ０において、オフセット６７８，１２３で見つかる。このリンクがＩＤ１，４５６，７８９を有していたと仮定する。最適ルートに沿って我々をこれに導いたリンクは、ファイルＴＴＢａｃｋＩｎｄｅｘ０において、オフセット１，４５６，７８９で見つかる。ファイルから抽出されたリンクＩＤがソース・リンクｌｉｎｋＡのリンクＩＤに合致するまで、我々はこれを繰り返し続ける。そうすると、我々は、最適（最も短い）パスのリンクＩＤの全てを逆順に抽出したことになる。一旦ダイクストラのアルゴリズムが終わってしまうと、これは最短パスを抽出する一般的な方法である。

【０１４２】

この抽出法は、活動ノードが全くなくなるまでルーティング・アルゴリズムを走らせたのであれば、ｌｉｎｋＡからグラフ中の他のいかなるリンクｌｉｎｋＣまでの最短パスを見つけるためにも機能する。ルーティング・アルゴリズムが早く終了された場合、我々はグラフ中のノードのサブセットへの最適ルートを見つけるために木を用いることができるだけである。

【０１４３】

＜ルーティング・アルゴリズム、目的地木＞

すべてのリンクから所定のリンク（目的地ルーティング木）までの最小コスト・ルートを造るために我々が用いるルーティング・アルゴリズム（ダイクストラのアルゴリズムのバージョン）は、以下の通りである。

【０１４４】

そのリンクから目的地リンクまで移動するためのそれまでに見つかった最小限のコストであるｍｉｎ－ｃｏｓｔを有するすべてのリンクから始める。ルートの目的地（目的地リンク）（ｍｉｎ－ｃｏｓｔがゼロである）以外のすべてのリンクについて、それを無限大にする。活動ノード（我々がまだそれらの入リンクの全てを探索し終えていないもの）の可変長リストを造る。最初はこのリストに目的地リンクデータのみを読み込む。

【０１４５】

これを繰り返す。

アクティブリストが空である場合、我々は終了したので、リターンする。そうでなければ、アクティブリンクのリストから第１のアクティブリンクを削除し、それをｌｉｎｋＱと呼ぶ。それは、我々が目的地に着くことができることがわかっている最小コストｃｏｓｔＱを有する。入リンクｌｉｎｋＮＱごとに、ｌｉｎｋＮからｌｉｎｋＱへ行く増分費用ｃｏｓｔＮＱを加える。そして、それが入リンクと共に格納されている最小コスト（ｃｏｓｔＮ）未満かどうか確かめる。もしもそうであるならば、ｃｏｓｔＮをそのより低い値に更新し、ｌｉｎｋＱを参照するためにｌｉｎｋＮのバックポインタを更新し、アクティブリンク）のリストに、ｌｉｎｋＮを加える（それがまだそこにない場合）。

【０１４６】

このアルゴリズムが終了するとき、我々はルーティング木を造り終えたことになる。

【０１４７】

＜最適ルートをたどる（トレースバックする）こと＞

目的地木を用いてソース・リンクｌｉｎｋＡから目的地リンクｌｉｎｋＢへの最適ルートを発見するために、我々は、ＴＴＢａｃｋＩｎｄｅｘ１ファイルを用いなければならないだけである。例えば、ｌｉｎｋＡがＩＤ　２３８，３４５を持っていたと仮定する。ｌｉｎｋＡが最適ルートに沿って先行したリンクは、ファイルＴＴＢａｃｋＩｎｄｅｘ１に

おいて、オフセット２３８，３４５で見つかる。このリンクがＩＤ　１，４３２，８７６を持っていたと仮定する。このリンクが最適ルートに沿って先行したリンクは、ファイルＴＴＢａｃｋＩｎｄｅｘ１において、オフセット１，４３２，８７６で見つかる。我々は、ファイルから引き出されたリンクＩＤが目的地リンクｌｉｎｋＢのｌｉｎｋＩＤに合致するまで、これを繰り返し続ける。そうして、我々は、最適（最も短い）パスのリンクＩＤの全てを正順（前進方向）に抽出する。一旦ダイクストラのアルゴリズムが終わっているならば、これは最短パスを引き出す一般的な方法である。

【０１４８】

なお、このパスは、上記セクションにおいて、ソース・ルーティング木から引き出されたものと同一となる。ダイクストラのアルゴリズムをソースノードから外側へと走らせるか、目的地ノードから内側へと走らせるかは重要でない。それはｌｉｎｋＡからｌｉｎｋＢまで単一の最短パス・ルートを常に見つける。しかしながら、ダイクストラのアルゴリズムの副産物であるルーティング木同士は、非常に異なる。それらが完全に共有する唯一のルートは、グローバルな最短パス・ルートである。

【０１４９】

この抽出法は、活動ノードが全くなくなるまでルーティング・アルゴリズムを走らせたのであれば、グラフ中の他のいかなるリンクｌｉｎｋＣからｌｉｎｋＢまでの最短パスを見つけるためにも機能する。ルーティング・アルゴリズムが早く終了された場合、我々はグラフ中のノードのサブセットへの最適ルートを見つけるために木を用いることができるだけである。

【０１５０】

＜ルーティング木の合成(combining)＞

一旦我々がソース・リンクｌｉｎｋＡからの木ＲＴａおよび目的地リンクｌｉｎｋＢまでの木ＲＴｂを計算してしまうと、我々は豊富なコスト情報も持つことになる。

【０１５１】

例えば、グローバルな最適ルートに沿ってｌｉｎｋＡからｌｉｎｋＢへ行くコストを見つけるために、我々は、目的地ルーティング木のための最小コストが入っているＴＴＭｉｎＣｏｓｔ１ファイルにおいて、オフセット＜ｌｉｎｋＡ＞を見る。

【０１５２】

我々は、ソース・ルーティング木のための最小コストが入っているＴＴＭｉｎＣｏｓｔ０ファイルにおいて、オフセット＜ｌｉｎｋＢ＞を見ることもできる。この値は、同じである。

【０１５３】

さて、我々が他のリンクｌｉｎｋＣに興味がある場合にはどうだろうか。ファイルＴＴＭｉｎＣｏｓｔ１におけるオフセット＜ｌｉｎｋＣのコスト＞でのコストは、ｌｉｎｋＣから目的地ｌｉｎｋＢへの最小限コスト・ルートのコストである。ファイルＴＴＭｉｎＣｏｓｔ０におけるオフセット＜ｌｉｎｋＣ＞でのコストは、ソース・リンクｌｉｎｋＡからｌｉｎｋＣへの最小限コスト・ルートのコストである。これらの２つの量を合計すれば、我々はｌｉｎｋＡからｌｉｎｋＣを経てｌｉｎｋＢまでの最小限コスト・ルートのコストを見つけたことになる。

【０１５４】

したがって、更なる反復なしで、我々は、ｌｉｎｋＡからグラフ内の他の任意のリンクｌｉｎｋＣを経てｌｉｎｋＢへ行くコストを見つけてしまった。我々は、ファイルＴＴＢａｃｋＩｎｄｅｘ０およびＴＴＢａｃｋＩｎｄｅｘ１に入れられているバックポインタを用いることにより、表示または他のために最適ルートを素早くたどることもできる。これは非常に強力な結果である。なぜならば、我々は（経由するグラフ中のリンクと同数程度の）代替ルートの全体集合をそれらのコストとともに効果的に計算したからである。もちろん、これらは可能なルートの全てのうちの小さいサブセットにすぎない。なぜならば、それらは、ｌｉｎｋＡからｌｉｎｋＣまでグローバルに最適なルートをたどり、ｌｉｎｋＣからｌｉｎｋＢまでグローバルに最適なルートをたどることを強いられているからであ

る。
【０１５５】
＜台地＞
いまや、我々は、ルートが唯一でないことに気がつく。いくつかのリンク、即ちｌｉｎｋＰおよびｌｉｎｋＱが、ある。そこにおいて、ｌｉｎｋＰを経由してｌｉｎｋＡからｌｉｎｋＢへ行くことは、ｌｉｎｋＱを経由してｌｉｎｋＡからｌｉｎｋＢへ行くのと全く同じルートを生成する。１のルーティング木（即ちソース・ルーティング木　ＲＴａ）におけるｌｉｎｋＰおよびｌｉｎｋＱ間のルートが他のルーティング木（ＲＴｂ）におけるｌｉｎｋＰおよびｌｉｎｋＱ間のルートと同一のときに、これは起こる。すなわち、ｌｉｎｋＰおよびｌｉｎｋＱ間の道の区画は、両方の木によって、連続して用いられたわけである。
【０１５６】
これらはまさに、我々が多様な代替ルートの一部として興味がある道の「良い」チェーンである。それらは、両方の木において、連続的に現れるチェーンである。我々はそれらを、リンクが入って離れる道を比較することにより見つけることを試みることができ、あるいは、ＲＴａおよびＲＴｂからのコストの総和を用いることができる。
【０１５７】
それらのチェーンは、最小コストルート上のｌｉｎｋＡからｌｉｎｋＢへチェーン内のリンクのいずれかを経由して行くコストが同じコストであることにより特定される。このために、我々はこのようなチェーンを台地と呼ぶ。なぜならば、それを構成するリンクに沿って、コストの総和が一定であるからである。
【０１５８】
次の２つのサブセクションは台地の一部であるリンクを見つけるための代替方式を提供する。それらの方法は、コスト関数の特性、および当該方法を実施するメモリおよびプロセッサアーキテクチャに応じて、異なる効果を持っているかもしれない。これらの方法が、異なるグラフ表示、異なるフラグ格納手段、および異なるコスト関数特性に適合できることも、明らかである。
【０１５９】
＜合成コストから台地リンクを見つけること＞
台地の全てを見つけるために、我々は、ｌｉｎｋＩＤｓの全てをスキャンする。それぞれのために、我々は、各木（ＲＴａおよびＲＴｂ）において、ＢａｃｋＩｎｄｅｘにより示されているリンクを見て、コストの総和が双方とも同じであるかどうか見る。それらが同じである場合、我々は台地の一部としてこのリンクにフラグを立てる。フラグは、通常通り、メモリマップ式ファイルに格納される。それは、ＴＣＦｌａｇｓ０１と呼ばれ、１リンクにつき８ビットを入れている。それは、この手順の初めに初期化されて、すべてのビット集合を０にセット（フラグの設定解除）する。我々は、そのオフセットを有するリンクが台地の一部として特定されたかどうかを示すフラグとして、最下位ビット（ｂｉｔ７）を用いる。
【０１６０】
フラグのファイル定義を示す。ファイル内の要素の数は、ファイル名の後、角括弧内に示され、その後には、各要素のタイプが続いている。
【０１６１】
ファイル記述：
ＴＣＦｌａｇｓ０１［ｎＬｉｎｋｓ］：０および１で表されるＲＴを合成するときに、フラグとして用いられる一連のＩＮＴ８。
一旦この走査が終了すると、我々はチェーン（ｂｉｔ７を用いる）の一部であるリンクの全てに、端の２つのリンクを除き、フラグを立てている。
【０１６２】
コスト値が整数である場合のみ、この方法は機能する。その結果、丸め誤差は我々が期待している同等性を損なわない。更に、木がいかなる順序で造られたとしても、木内のリ

ンクを辿るためのコストが常に同じである場合のみにこの方法は働く。これは大概そのとおりであるが、例えば、リンク・コストが時間に依存するものであって、２本の木において、わずかに異なる時間に辿られる場合、この方法は機能しないだろう。したがって、バックポインタに基づく代替方式がその代わりに用いられなければならない。

【０１６３】

＜バックポインタから台地リンクを見つけること＞

台地の全てを見つけるために、我々は、すべてのｌｉｎｋＩＤをスキャンする。ｌｉｎｋＩＤの値、即ちｌｉｎｋＰごとに、我々は、そのリンクがソース木において、バックポインタを持っているかどうか見る。ソース・リンク以外のすべてのリンクは、有効ポインタを持っている。それがｌｉｎｋＱを指していると仮定する。そうすると、我々は、オフセットがＩＤ　ｌｉｎｋＱであるＴＣＦｌａｇｓ０１ファイル内のバイトに指名ビット（即ちＴＣＦｌａｇｓ０１ファイルにおけるｂｉｔ５）をセットすることによって、ｌｉｎｋＱをソースポインタを有するものとしてマークする。我々は、ｌｉｎｋＰが目的地木において、バックポインタを持っているかどうかも見る。もしそうならば、つまりｌｉｎｋＲならば、我々は、オフセットがＩＤ　ｌｉｎｋＲであるバイトに、異なるビット（即ちＴＣＦｌａｇｓ０１ファイル内のｂｉｔ４）をセットする。我々がｂｉｔ４およびｂｉｔ５でフラグのどちらかをセットする場合、我々はもう一方を見て、それもセットされている場合には、このリンクが台地の一部であることがわかるので、我々はコスト方法と同様に台地ビット（ｂｉｔ７）をセットすることもできる。

【０１６４】

一旦この走査が完了すると、我々はここでも、チェーン（ｂｉｔ７を用いる）の一部である全部のリンクに、２つの端リンクを除いて、フラグを立てたことになる。

【０１６５】

＜台地リンクを集めること＞

我々は、今度は、チェーン自体を見つけて、それらの特徴を描写することが必要である。我々は、新たにリンクの全てを走査することによりこれを行う。

【０１６６】

リンクごとに、我々は、（ｂｉｔ７を用いて）それが台地の一部としてフラグを立てられたかどうか見るために調べる。そうであった場合には、我々はｂｉｔ６を見てそれがすでに処理されたかどうか見る。そうでなかった場合には、処理されたものとしてそれにマークし（ｂｉｔ６を用いて）、遭遇するリンクが台地の一部としてマークされる限り、上記ソース木および目的地木内をたどる（トレースバックする）。我々はすべてのこのようなリンクをビット６を用いて処理されたものとしてマークするので、走査の後段階でそれらに再び遭遇しても、この台地では我々はどのリンクも処理しないだろう。

【０１６７】

このようにして台地を追跡するので、我々はそれ以後の使用ために台地についての情報、たとえば、その長さおよびそれを辿るコスト等の情報を集めることができる。しかし、この特定の実施例では、我々は最良のプラトー選択の基礎としてコストを用いるだけである。そのコストはＴＴＭｉｎＣｏｓｔファイルにすでに保存されているので、我々はそれらを再度計算する必要がない。むしろ、我々は、台地の端のｌｉｎｋＩＤに注意する。ｌｉｎｋＩＤｐｓはソースに最も近い台地の端である。そして、ｌｉｎｋＩＤｐｄは目的地に最も近い台地の端である。我々がこうして見つけた台地ごとに、（可能ならば）我々は、１つのリンクを上記端からソースおよび目的地近傍まで延ばし、ＲＴａまたはＲＴｂのいずれかから調べられたそれらのリンクについて最小コストの差を計算する（台地が各木の接続部を形成しているので、結果は同一でなければならない）。これが、我々が台地を辿るためのコストとするものである。

【０１６８】

＜良好性の基準＞

さて、スキャンするとき、我々は一般に、より短い台地を拒否し、より長いものを保つことを望む。また、我々は、あまり長くないルートの部分を形成する台地を支持すること

を望む。我々がケンブリッジからマンチェスターへ行く場合、我々は多分、スコットランドにある長い台地（それは道のある配列により時々形成される）には興味を起こさないだろう。

【０１６９】

より高い関心のある台地だけを選択するために、我々は「良好性」値を計算する。我々は我々が興味がある台地に対してそれがより高い値を持つようにアレンジしている。そうして、我々は、最も高いｎ個の良好性値を有する台地の端部リンクを記録する。ここで、ｎは、我々がおそらくユーザに表示したくなり得る数（即ち１００）より大きい。良好性は、ルートの特性（例えば距離、財政的コスト、道のタイプ、ソースおよび目的地からの距離）のいずれかを用いて、いかなる方法でも計算されることができる。

【０１７０】

この実施において、我々は、（最小コスト（ｍｉｎｃｏｓｔ）を計算するために用いられるような）コストに良好性値の基礎をおくことを選ぶ。我々は、端間のコスト差がより大きい台地に対してより高い良好性を欲する。というのは、それらは、一致させられたルーティング・ツリーのより長い部分を表すからであり、それ故、ｌｉｎｋＡからｌｉｎｋＢまで行くのに局所的に役立つより長く伸びた道を表すからである。しかしながら、我々はまた、全体コストがより高いｌｉｎｋＡからｌｉｎｋＢまでのルートを形成する台地に対してはより低い良好性値が欲しい。

【０１７１】

したがって、単純な良好性値は、台地を辿る全体コストから、この台地を用いるｌｉｎｋＡからｌｉｎｋＢまでのルートのコストを引いたものである。これは一般に負（マイナス）であり、つまり、台地自体の中で含まれない、台地を用いたルートのコストの否定である。この為、我々は、ルートが台地の外側に含まれる最小コストを有する台地を優先して保持している。

【０１７２】

我々が台地の各走査を完了したとき、我々はその良好性を計算し、我々がソート済みリスト内で見つけたトップｎ個（通常は１００個）を保存する。ある台地の良好性が上記リストにおいて、最も低い良好性を有する台地より大きい場合、その台地の詳細を保存しさえすればよいことを我々は知っており、我々はそれに置換する。

【０１７３】

＜最良の台地選択＞

我々が定義した良好性値は、台地を、我々がそれらをユーザに示したい順番に置いてしまっているが、我々は幾つを示さなければならないのだろうか。

【０１７４】

これを行うため、我々は、良好性値を、ソースおよび目的地リンクならびに道グラフのコストおよびスケールから独立しているスケールで台地を評価できる何かに変換することが必要である。これを行うため、我々は、まず最初に、グローバルに最小限のコスト・ルート（ＧＭＣｏｓｔ）のコストで良好性を割ることによって、良好性を無次元とする。これを「生の良好性」(raw goodness) と呼ぶ。

【０１７５】

つまり、生の良好性とは、台地の一部でない台地を伴うルートのコストを、グローバルに最小コストのルート（ＧＭＣｏｓｔ）のコストで割った値の負数である。

【０１７６】

台地外にはゼロコストがあるので、この生の良好性は、グローバルに最小限のコスト・ルートに対してゼロの値を有する。

【０１７７】

では、２つの台地は決して互いに接触することができないため、グローバルに最小限のコスト・ルート（それはそれ自体がその全長に対しての台地である）からのいかなる小さいずれも、小さい台地を含むことができるだけである点を考えてみよう。小さいずれの外側にあるコストは、ＧＭＣｏｓｔよりわずかに低いだけであり、したがって、生の良好性

は－１よりわずかに大きい負数（例えば－０．９９４、－０．９８７、－０．９９２など）である。

【０１７８】

では、辿りコストがＧＭＣｏｓｔと粗方同一である台地はどうであろうか。それがソースおよび目的地リンクの非常に近傍まで延びている場合、それはそれ自身の外でほとんどコストを持っていない。したがって、生の良好性は０よりもわずかに負で、おそらく０．０２７、０．０１２、０．０２３などである。台地の外側のコストがＧＭＣｏｓｔに匹敵するほど台地がソースおよび目的地リンクから十分に遠い場合、生の良好性はおよそ－１である。台地がより近くであって、台地の外側のコストがＧＭＣｏｓｔの半分である場合、それは－０．５の生の良好性を有する。

【０１７９】

これらの値をよりわかりやすくするために、我々は、単調にそれらを他の範囲に変換する。そこにおいて、０（最良の）は９９に位置し、－０．５は９０に位置し、－０．８５は５０に位置し、そして－１はゼロに位置する。これは、ユーザへの表示のために用いられるだけであり、約２０と９９の間の値が「不良」から「良好」までのルートの範囲にわたっている。

【０１８０】

用いられる関数は、次の通りである。

$$良好性＝１００－（９９^{－生\_良好性}）$$

【０１８１】

考慮に値する多様な代替ルートの最小の集合を自動的選択するために、我々は良好性分離（カットオフ）点を５０（または生の良好性で－０．８５）に設定し、表示装置やその他のものを介してユーザにこれらの選択だけに気づかせる。

【０１８２】

ケンブリッジからマンチェスターについて、これは６つのルート（我々のテストデータベースは、Ｍ６料金よりも前のものである）だけを選択する。

【０１８３】

＜成功の基準＞

成功と考えられるために、我々の方法は、合理的に思えるルートを見つけ出さなければならないだけで、異様なものを見つける必要はない。我々はこれを達成するために旅の時間または旅の長さに制限を加えることができた。しかし、少なくともこの出願において、上で概説された良好性要因はこの予備のステップを不必要とするように思われる。

【０１８４】

多様な代替ルートの集合は、我々が自分自身の経験から良好なルートとして知っている全てのものまたは少なくともそれらの軽微なバリエーションを含むべきである（なぜならば、ルーティング・データベースはコストについてのユーザの考えを完全には反映していないが、より長い伸びにおいては粗方正しいからである。）。多様な代替ルートの集合は、他のルート・プランナ（例えば（英国用に）マイクロソフト社のオートルート、アルク社のコーパイロット、メディオン社のナビゲータ、アイシス社のパーソナルナビゲータ）により計算された最良の単一ルートの各々を含むべきである。

【０１８５】

ソースおよび目的地が数百マイルも離れていようが１０マイルしか離れていまいが、上で概説した方法が、これらの基準を満たす一組の多様な代替ルートを産生することを我々は発見した。ソースおよび目的地がかなり近接している（即ち１マイル）場合、ルートはあまり多様ではあり得ないので、ほぼ同一の長さであるが多様性の良好なほんの少数のルートだけが提案されるかもしれない。それらが非常に近接していて１つのルートのことしか思いつかないような場合には、１つのルート、グローバルに最小コストなもの、だけが提案される。もちろん、多くの他のものが我々の方法により生成されていたであろうが、１つのルートだけが５０を超える良好性を有するだろう。

【０１８６】

＜台地アルゴリズム・コスト＞
台地を見つけ、分類し、ソートする際に伴う走査はダイクストラのアルゴリズムまたはその異型を実行するより一般に少なくとも１００倍速いので、この全てのフェーズは、単一の最適ルートを見つけることよりもかなり少ない時間しかかからない。したがって、我々の方法の全体のコストは、単一ルート－ファインダーを２回走らせるコストにその何分の１かを加えた程度である。大ざっぱに言って、単一の最短パス・ルートを見つけるよりも２～３倍の時間かかるが、我々は、何千ものものを考慮し、そして、ユーザに表示するために最良のわずかなものを選択している。

【０１８７】
＜他の基準の使用＞
我々の実施例は、時間（秒）の１００倍プラス長さ（メートル）であるコスト関数を用いる。したがって、我々が１ｋｍを余分にドライブすることによって、旅（６０秒）の１分を節約することができれば、我々は各ルートをほとんど均一であると思うだろう。これは、良い選択肢の集合を生成するための良い一般的設定のように見える。それは、最も速いルートの方ほど重く重み付けをしている。

【０１８８】
より短いルートを支持するために、我々は時間（秒）の１０倍プラス長さ（メートル）とするかもしれない。そして、それはほぼ同一の選択肢を与える。

【０１８９】
コスト関数として純粋な距離を用いることにより完全に時間を無視しても機能し、我々は異なる組の選択肢を得る。ほとんどの場合、ルートの全てはかなり遅いが、我々はなお同様のコスト（このケースにおいて、距離）の選択肢の多様な集合を得る。

【０１９０】
単一ルーターのために用いられるかもしれない他のものをコスト関数に含めてはならない理由はない。これらは、財政的なコスト（ガソリン、消耗品、道路使用料金制、通行料金）、歴史的なおよびリアルタイム渋滞情報、安全性、天気、道のタイプ、回避する領域、ドライバの精通性、その他多くのもの）を含む。旅が進むにつれて、これらのいくつかは変化する。そして、我々はそれらの新しい値に適応したいと思うかもしれない。

【０１９１】
例えば、車両に供給されている最新の渋滞情報に従ってルート選択が行われるシステムを記述するために、フレーズダイナミックルーティングが用いられる。このような情報を用いるために、我々は、データベースにおいて、渋滞その他の要因によって、影響を受けるリンク上の遅延を更新し、我々の方法を再実行することができた。これは、新情報が利用可能なときに単一ルーターがすることである。

【０１９２】
我々は別の、もっと興味深い利用可能なアプローチを有する。我々がすでに計算された良い多様なルートの集合を持っているので、我々は単一ルーターを動かすのに必要である時間のうちのほんの少しの時間で上記新しいコスト基準に従ってそれぞれのルートを評価できる。そして次に、我々は、ユーザに表示するためにこれらの新しいコスト基準を用いてルートの順番を付け直すことができる。

【０１９３】
我々は、一日の異なる時間にコストおよび期間について多様なルートの各々を点検し、ユーザに対して、彼らの旅をするのに最適な時間あるいは最適な日さえ提供できる。これがあらゆる曜日（１０００回）の１０分ごとに単一ルーターを再実行することによって、されるならば、とられる時間は極端に大きくなるだろう（４０００秒＝６６分）。最初に多様なルートの集合を選択し、それから時間依存コスト・パラメータを用いてそれらを評価する我々の方法は、わずかな時間おそらくこのスケールで約１２秒で動作する。それはむしろ、人間がルート選定マップを見、道タイプ（速度）および方向に基づいて、用いるべきありそうなルートを選択し、そして、すべてのコストを合計するという激務をするためにコンピュータの助けをかりて、それらのルートをその特性について評価することに類

似している。これはコンピュータにとって、それが最少のコストを見つけ出すためにすべての可能な最適ルートのコストを評価しなければならない場合に比べて、はるかに少ない仕事である。

【０１９４】

動的な情報のなかには大きく再計算が必要であるあるものがある。したがって、我々は我々のシステムを既存の技術への追加とみなす。例えば、他のあらゆる多様なルートが現在のルートを離れる前に、大きい遅延を前もって示す新情報が旅中に入った場合、我々はこの新しい障害物周辺で単一ルートを計算するために既存の技術に頼るだろう。もちろん、一旦これがなされると、我々は続行して我々が現在いるところから目的地までの多様なルートを計算し、必要に応じて選択肢を提供できるだろう。

【０１９５】

この選択ルーティングの重要な特徴は、それがユーザに伝える情報である。単一ルーターが動く多くのケースにおいて、ユーザは、如何に最適ルートが彼らが選んだかもしれないルートに匹敵するかについて理解しない。実際、ほとんどの場合、両方のルートは、非常に類似したコストを有するだろう。我々がユーザに相対的な時刻および距離を示せば、彼らは自分が好むルートを用いるか、新しく推奨されたルートを取るかの選択を行うことができる。我々が回避したであろうものは、ユーザが何か間違ったことをしたかもしれないという不確定性である。それは、例えば、間違った目的地の選択、間違った好みの設定等である。提案最適ルートが、普通でない渋滞のために行われたのであれば、これは、選択点（極めて少ない）で、第一の最善ルートおよび今日推奨されるルートの両方のオプションをそれぞれの旅時間情報と共にユーザに示すことによって、ユーザに明らかにできる。実際、我々は、旅時間が通常のものとは著しく異なる場合には、強調さえするかもしれない。

【０１９６】

例えば、Ｍ６を伴うルートが２０分余分に遅延を被ると仮定する。これは、Ｍ１へのルートを約５分早めることになるかもしれない。単一ルーターはＭ１ルートの方へとユーザを自動的に案内するが、我々の方法は彼らに、（彼らが他の理由のために好むかもしれない）Ｍ６上にとどまって５分を犠牲にするという、情報に基づいた選択を行わせるだろう。

【０１９７】

この情報は、音声プロンプトまたは模式的なジャンクション表示（道路標識のようなもの）といった一般的な形で与えることができる。我々のシステムについての差は、旅の重要な部分において、ガイダンスが、たどるべき単一のルートを単に提示するのではなく、２つ以上のオプション（２を超えることはめったにない）を、それらの相対的な旅時間に関する情報と共に提示することである。これの例は、図１５に示される。

【０１９８】

＜パフォーマンス＞

ルーティング計算の速度を上げるために、我々は、グラフにおいて、最近選択されたリンクへの、そして、それらのルートからのルーティング木を保存しておくことができる。これは、簡単なリコール（再呼び出し）のための最近入力されたロケーション（時々「好きなもの」リストと呼ばれている）の保存の拡張である。ルートの一端がリストに載っている場合、我々はそのルーティング・ツリーのどちらも計算する必要はない。他のいかなるノードへの、あるいは、からの最良の単一ルートは、木をバックトレースすることによって、すぐ見つかる。ルートの両端がリストに載っている場合、我々は我々の選択肢ルーティングのために必要である両方のルーティング・ツリーをリコールすることができて、もはやダイクストラのアルゴリズムを走らせることを必要としない。したがって、我々は、従来のルーターが単一の最善ルートを計算できるより速く、多様な代替ルートを見つけ出すことができる。

【０１９９】

それらの木は、それらのコストおよびバックポインタが計算済みのファイルを保持する

ことにより保存されることができる。あるいは、代替実施例が用いられたのであれば、新規なファイルフォーマットをこの情報の記憶のために定義できるだろう。記憶が非常に高額であれば、バックポインタかコストのうちの一方だけが保存されることを必要とする。なぜならば、ルーティング・データベースの走査を用いて、もう一方を再構築できるからであり、ルーティング・アルゴリズム全体を走らせる必要はない。

【０２００】

＜時間依存ルーティング＞

リンクの速度が時刻によって、変化するかもしれない所では、または、道路使用料金が時間とともに変化する所では、または、リアルタイム交通情報が利用できる所では、従来のルーターはダイクストラのアルゴリズムと類似のものを走らせて最良のルートを見つける。その場合、リンク・コストは、最新の時間依存情報を反映するために変えられてしまっており、それらは、リンクが辿られるのに予想される時間に対して高感度である。

【０２０１】

我々の技術はまさに同一方法において、用いられることができ、最新のコスト情報を用いてダイクストラのアルゴリズムを２回走らせる。グローバルに最適なルートよりかなり長く時間がかかる選択ルートは、予想辿り時間よりも僅かに早いか遅い時間のために時間依存情報を用いるかもしれない。その理由は、もはやソースから目的地までの固有の旅時間はなく、むしろ小範囲の旅時間があるからである。これは問題となりそうにならない。いずれにしろ旅時間および予測された交通レベルには何らかの固有の(つまり内在する）可変性があるからである。

【０２０２】

予想辿り時間におけるこの不確実性を取り除くために、我々は生成した少数の最良ルートを取り、それらを個々に時間変動コスト・パラメータに対してすばやく再評価し、可能な限り最も正確な結果を得る。

【０２０３】

これは、更なる技術を提案する。第１のルーティングの実行において時間依存コスト・パラメータを用いる代わりに、我々は、基本的なコスト関数（例えば、時間（渋滞のない）および距離だけ）を用いることができた。一旦我々がこの基本的なコスト関数を用いて少数の選択ルートを見つけると、我々は、新しい時間依存基準に対してそれらの各々を再評価して、ユーザに提示するためにそれらを並べ替えるだけである。このようにして、我々は、変化に迅速に反応することができ、ユーザに対してなぜ別のものでなくこのルートが選択されたのかを明らかにでき、しかも、最善のルートとほぼ同程度に良い選択肢をユーザに示すことができる。

【０２０４】

コスト関数が時間依存または他のコスト・パラメータにより徹底的に変化した場合、制限されたコスト関数の下で計算された選択ルートは、完全なコスト基準の下では、グローバルに最適なルートをもはや含まないかもしれない。この場合、我々は、制限されたコスト関数および完全なコスト関数の両方のモードで、我々の選択肢ルーティング・アルゴリズムを走らせることができる。次に我々は、各モードからの最善のルートをユーザに表示することを選ぶことができる。したがって、彼らは、現在利用可能な非常に良いルートと共に、それらが、彼らが時間と距離だけで選んだかもしれないものにどのように匹敵するかを見る。

【０２０５】

完全な時間依存コスト関数の使用は、いくつかの状況において、逆効果でありえる。ひどい渋滞の下で、時間依存最適ルートは、交通センサも交通報告もないマイナーな道を巻き込む可能性が高い。これらの道は、流れるように進むように見えるが、実際は、それらは渋滞するかもしれないので、選ばれたルートは結局最適でないかもしれない。したがって、我々は、ユーザが望む場合ユーザが主要ルート上にとどまることを選ぶことができるように、時間依存情報なしで計算されたが時間依存情報によって、再評価された選択ルートの少なくともいくつかを表示することを常に推奨する。これはまた、救急隊が主要ルー

トを移動させ続けて機能停止に対処するような悪天の場合には、より安全なオプションでありえる。

【０２０６】

＜他のコスト要因＞

他のコスト要因が変化するとき、基本的なコスト関数を用いているルートは再計算される必要はない。我々が多様な代替ルートの各々を再評価するときに、ユーザに提示するためのルートの新たな順序を我々に与えるべく我々はコスト関数において、新規な重みおよび道リンク上の新しいコスト要因を用いることができる。実際、ユーザが彼らのオプションを探索するとき、良いデスクトップ・インタフェースは、ユーザが時間、距離、財政的なコスト、道のタイプ、曲折の数その他によって、代替ルートをソートすることができるようにするはずである。これはいかなる再計算も必ずしも必要とするというわけではない。なぜならば、それらの要因は、計算され、最初に生成されたとき代替ルートとともに格納されることができるからである。

【０２０７】

道路利用者課金がより一般的になったとき、役立つコスト関数を定義することが可能かどうかは、まだ知られていない。問題は旅のコストと速度の間のトレードオフが目的地での時間厳守の重要性（それは予測不可能な方法において、日々変化する。）に応じて変化するかもしれないということである。トレードオフは、如何に遅くドライバーが彼らの旅に出発したかに応じて変化することもある。彼らが早い場合、彼らはより速いルートに費用を払うことを望まないかもしれない。しかし、彼らが遅く走っている場合には、料金はより受け入れられやすいかもしれない。

【０２０８】

基本的なコスト・パラメータ（時間および距離）だけを用いて選択ルートを計算する我々の技術によって、我々はドライバに、賢明なルートの選択肢をそれらの推定時間およびコストとともに見せることができ、ドライバは、ルート・プランナが決して知っていないかもしれない予備情報の全てを考慮して、自身のための決定を行うことができる。これは、時々「人間をループに入れる」と呼ばれ、結果として生じるシステムをより簡単でより使い勝手のよいものとすることができる。なぜならば、ユーザがそれ自身のための複雑なトレードオフを行い、その一方で、コンピュータは旅時間、距離、コストその他を試算する激務をしているからである。

【０２０９】

＜交通計画(Traffic planning)＞

選択ルートは、単独で用いられる必要はない。我々が、旅の起点および目的地の形で旅情報を持っている場合、我々は選択ルートを用いて、これらのルートに用いられるリンクの全てに、ルートの良好性に比例して交通を割り当てることができる。良好性計算において、用いられるパラメータの調整によって、実際のデータに対して調整されるとき、そのようなツールの中心で選択肢ルーティング・アルゴリズムを使用することは、流れまたは単一ルート・アルゴリズムに基づく現在のツールを用いるよりも交通パターンのより正確な予測を与えるかもしれない。

【０２１０】

このタイプの方法は、例えば建物、航空機、プリント回路基板および集積回路において、ワイヤ、導線、導体またはケーブルのルーティングにも適用されるかもしれない。ワイヤのためのルートを見つけるときに、道ルートのためのケースとは違う点が２、３ある。特に配線ネットのルーティングおよび配置順序の付加的変数である。

【０２１１】

＜配線ネット＞

ワイヤが始まることができるいくつかの点およびいくつかの終端点（その内のどれも充分な目的地である）があるかもしれない。より複雑なのは信号ネットで、ワイヤは、いくつかの点集合の各々からの少なくとも一つの点に接続する必要がある。我々はこれらの点の全てを基点もしくは原点と呼ぶ。なぜならば、通行があるときの流れの方向がなく、起

点とか目的地とかの明白な概念がないからである。ダイクストラのアルゴリズムに類似の技術がしばしば用いられ、おそらく、同時にいくつかの起点から始まり、辿られる各ノードでの最小コストおよびバックポインタの通常の値を保持し、起点の全部が合成されると最適ルートを終了して、トレースバックする。これらの技術の多くのバリエーションがあるが、それらが各ノードで最小限のコストおよびバックポインタの基礎をなす構造を持っている限り、我々は我々の技術を適用して各起点ノードからの別々のルーティング木を合計し、同一のコスト－総和（台地）のチェーンを見つけ、いくつかの代替物選択ルート（または信号ネット）を、それらをたどって起点まで戻ることにより、生成する。

【０２１２】

＜配置順序＞

道ルーティングおよび配線ルーティング間の重要な差は、配線に対しては、我々は、通常、単一ワイヤのパスを最適化することに興味はなく、ワイヤ全体の集まりのパスを最適化することに興味をもっているということである。これは、我々がどのような順序で個々のルートのルーティングを行うようにすべきかの問題を生じさせる。ここでも、多くのバリエーションがあるが、一般的な技術は、ルートを見つけ、全ワイヤのうちのいくつかの位置を固定（ワイヤの配置」と呼ばれる）することである。そして、更なるワイヤが過密なパス等の問題を持っている場合、我々は配置済みのいくつかのワイヤを引き上げ、それらを未配線ワイヤのリストに加え、新しいワイヤをルーティングして、続ける。このリップアップ（切り取り）および再ルートの技術は、計算機的に非常に高価で、しかも非常に劣ったルートで終わるかもしれない。さまざまな周知の増強手段は、どの信号ワイヤが配置するのが最も困難になるかを、それらの長さ及びそれらの大雑把なパスに沿ったルーティング・チャネルの幅をみることによって、評価しようとする。あるいは、渋滞が起こったとき渋滞の領域を観測し、それらに対してコスト関数を重み付けをすることによって、それらがさらに渋滞するのを回避しようとする。

【０２１３】

＜配置順序のための選択肢ルーティング＞

選択肢ルーティングの我々の技術は、ワイヤが配置される順序を選ぶための重要な要因を提供できる。ワイヤが配置される前に、ワイヤごとに選択ルートを計算することによって、いずれが利用可能な良いルートを１つだけ（道の場合において、ケンブリッジからカンタベリーへのようなもの）持っているか、そして、いずれが幾つかの良いルート（ケンブリッジからマンチェスターへのようなもの）を持っているか見ることができる。最初により少しの選択ルートを有するワイヤを配置することは意味をなす。なぜならば，それらが後のワイヤのためのルートのいくつかをブロックすれば、それらの後のワイヤは利用可能な他の良い代替物を有することになるからである。更に、我々が幾つの良好なルートが利用可能かということのみならず、それらが如何に良好であるかを参酌するよう、我々の良好性値を用いることができる。典型的実施は、配線ネットごとに、閾値（即ち２０の最小良好性が２０）を越えるトップ１０のルートの良好性値を合計し、最も低い総和を有するルートを最初に配線する。もちろん、この技術は周知の技術に置き換わるものでない。というのは、我々はなお、いくつかのネットをリップアップ及び再ルーティングしなければならないかもしれないからである。しかし、我々は、利用可能な選択肢の量によって、未配置ネットをソートして、最少の選択肢を有するものを常に最初に配置する。それは、解法を見つけるための時間の速度を上げるはずで、さらに重要なことに、配線ネットを全体的により短く保つことによって、または、このような広いルーティングチャネルを要求しないことによって、最終結果を向上させる。

【０２１４】

図１７は、ナビゲーション装置の例を示す。この装置は、プログラムメモリ１１（例えばＲＯＭ、フラッシュメモリ、ハードディスクドライブおよび／または光学ディスクドライブ）を有するコンピュータ１０の形態をとり、車両（例えば自動車）に搭載される。しかしながら、このような装置は、車両で用いるものに制限されなくて、他の物体に取付または搭載してもよい。例えば、このような装置は、例えばその地理的場所および現在時刻

に関する情報を受信するタイプの携帯または移動電話に搭載できる。

【０２１５】

車両は車両電子システム１２を備える。そして、車両電子システム１２は車両に搭載された装置を、多く監視し制御する。コンピュータ１０は、したがって、車両の現在の状態のさまざまな面に関する情報を受信することが可能である。

【０２１６】

車両は、アンテナ１４を介してＧＰＳから受け取る信号から車両の位置および現在時刻を決定するためＧＰＳレシーバ１３を更に備える。例えば衛星航法機能または「サテライトナビ（Sat Nav）」システムを提供するために、この情報は車両電子システム１２にも供給される。

【０２１７】

コンピュータ１０は、人間の知覚できる出力を提供するために出力装置１５も備える。出力装置１５は表示装置として示されており、例えば、サテライトナビシステムまたは車内娯楽システムの一部を形成しているが、それに代えまたは追加的に、スピーカ等の音声出力装置を備えてもよい。

【０２１８】

コンピュータ１０、プログラムメモリ１１およびレシーバ１３は図１７において、別々の部材として示されているが、それらは、ディスプレー１５も含む単一の装置として具体化されるかもしれない。

【０２１９】

＜道タイプを用いた制限検索＞

より速くルート計算を行う一般的な手段は、検索を道のサブセットに制限することである（それらは、旅の起点または目的地から十分に遠いときには、一般に、より速い、あるいは、より有用である）。我々はそれらを優先道と言う。そして、サブセットはそれらが完全に接続されることを確実にするために、ジャンクション、進入路その他を含むように調整される。このような道の集合は、図１８に示される。ソースおよび目的地木が各木において、同じ方向に辿られる道リンクのチェーンを見つけるために合成される場合、選択肢ルーティングの技術は、このような制限された木によってもうまく機能する。しかしながら、選択肢ルーティングの場合、それは、両方の木において、起点および目的地で遅いローカル道をカバーすることは必要でない。

【０２２０】

ソース木については、我々はソースから外方へ捜すので、いくつかの限度に達するまではすべての道タイプを用いることができ、その後、より速い道を考慮することに切り替えるだけである。ソース木の例は図１９に示される。上記限度は、速い道が最初に見つかるコストに注目し、そして、予備のコスト（時間および距離において、）およびおそらくパーセンテージ超過を許容することにより良好に決定される。例えば、時間だけを用いると、我々は、２０％の超過プラス１０分を用いるかもしれない。我々が起点から１３分のところで第１の速い道に遭遇するならば、最小限のコスト面が（１３＊１．２）＋１０＝２５．６分に達するまで、我々はローカル道が用いられるようにするだろう。我々がより段階的な道分類法を用いるならば、我々はいくつかの閾値、例えば、ローカル道検索、都市道検索、都市連絡道検、幹線道路検索のための閾値を持つことができるだろう。

【０２２１】

ソース木については、それが目的地に接近するとき、道タイプの従来の使用は、我々が近付いている（つまり、５０ｍｐｈの速さで直線距離にして３０分以内のところ、すなわち、２５マイル以内のところである）ことを検出し、再び遅い道タイプを検索することへ切り替えるためにこれを用いるだろう。優先道のいずれも目的地の２５マイル以内に来ないかもしれないので、更なる問題がここにある。したがって、我々はローカル・ルーティングへ決して切り替わらないかもしれないので、我々は完全に目的地を見逃すだろう。これを避けるために、もう少し高度な方式を用いて、いくらか大きい範囲でローカル・ルーティングへ切り替わるようにすることができるだろう。これは従来行われていることで、

そうすると、選択肢ルーティングは上記ソース木および目的地木を通常の方法で組み合わせることができる。

【０２２２】

しかしながら、選択肢ルーティングにおいて、我々は、より良いオプションを持っている。それが台地の小さい一部にすぎないので、我々はソース木が目的地に着くこともその近傍において、ローカル道を探索することも必要としない。この為、我々はソースの近くにあるすべてのローカル道を探索するソース木を持っている、しかし、我々がソースから移動するにつれて、探索する道タイプはすくなくなり、目的地が速い道タイプの１つにない場合、恐らく目的地を見失う。ここで、我々は、同じように目的地木を計算するが、今回は、目的地の近くにあるすべてのローカル道を探索する。そして、目的地から移動するにつれて探索する道タイプは少なくない、ソースが速い道タイプの１つにない場合、恐らくソースを見失う。その例が図２０に示される。次に、我々は、我々の通常の方法でこれらの木を合成する。ソースと目的地との間に有用に整列配置される良い速い道について、我々は、それらが両方の木に存し同じ方向に辿られるので、台地が通常通り形成されることがわかる。その例が図２１に示される。１つの違いは、台地が必ずしもソースから目的地までずっと伸びているわけではなくて、最も長いものが各先端でぴったりと止まるということである。これは、ローカル道が両方の木で探索されたわけではなかったためである。但し、これは問題でない。というのは、我々は最初に最も長い台地を見つけ、そして、通常の方法で最適ルートを生成し、ソース木（それは、ソースの近くでローカル道を用いる）を用いてソースまでたどり、目的地木（それは、目的地の近くでローカル道を用いる）において、目的地までたどるので、各先端でローカル道を用いる完全なルートを得るからである。その例は、図２２に示される。

【０２２３】

この方法は、ソースおよび目的地で用いられるより遅い道を有するソース木または目的地木を計算しようとする方法に勝る２つの利点を持っている。第１に、我々は各木において、２つの遅い道タイプ検索から各木のための１つだけに計算諸経費を減らすので、選択肢ルーティングはより効率的であり、従来のルーターを動かすだけのコストに近くなる。第２に、ソース木において、起点から、または、目的地木において、目的地から離れて移動するときに、いつ遅い道からより速い道タイプへ切り替えるべきか決定がかなりより容易である。これは、我々が、どれくらいの数の高いタイプの道に我々が遭遇したかに気がつくことができ、スイッチング基準の一部としてそれを用いることができるからである。これは、より速い道タイプの階層のために、数回、容易になし得る。ソース木において、目的地の方へ進むとき、従来のルーターはより高い諸経費がかかるだろう。これは、より遅い道タイプが最後の３０マイルに対して用いられなければならない困難なケースが随時あるからであるが、それは事前にはわからない。この為、従来のルーターは、常に、この最悪のケース距離で切り替わるであろう。しかし、遅い道タイプは最後の５マイルについてだけ必要かもしれない通常のケースに対して、３０マイルの距離でのより遅い道タイプへのスイッチングは非常に無駄である。

【０２２４】

＜貫通ルート木における時間依存性＞

我々が時間に依存する情報（例えば各日の５分間毎の平均道速度）を持っている場合、我々はまだ我々のソース木および目的地木を計算することができ、通常の方法において、それらを合成できる。例えば、我々が到達時間（すなわち目的地に達する時刻）を与えられる場合、我々は、コスト関数を推定するまさに同じ方法で各道リンクを辿る時刻を推定して格納し、その時刻に対する平均速度を用いて各リンクを辿るためにかかる時間およびコストを計算して、目的地木を最初に計算するだろう。一旦目的地木がソースに達してしまうと、我々はソースからの必要な出発時間の推定（見積もり）を持つことになるだろう。そして、我々はこの出発時間を用いてソース木を造ることができるのであるが、ここでも、その時刻に対する平均速度を得て交通速度における変化を考慮するために、各リンクに遭遇する推定時刻を用いる。我々は、それから、我々の通常の方法でこれらの木を合成

して台地を得て、我々の良好性関数に従って等級づけを行うことができる。これは適切に機能するが、もっとよくなり得る。問題は、グローバルに最適なルートにない特定の一連の道路区分は、それらの辿りについての時間見積がソース木において、目的地木におけるのとはわずかに異なる時間にあるということである。これによって、取るルートに軽微なバリエーションが生じるかもしれない。そして、それは、結果として生じる台地を２つに分け、旅全体とは無関係である軽微なバリエーションのために、その台地の良好性基準のランクを下げ得る。

【０２２５】

これらの軽微なバリエーションを回避するために、我々は、我々が計算する第１の木から、辿り推定時間を固定する。すなわち、道リンクごとに、我々がそのリンクに達したときのコストだけでなく、それを辿る推定時間も格納する。次の木を計算するときに、我々は、新たにそれらを計算するよりはむしろ、それらの時間を用いる。このようにして、各リンクを辿る時間依存コストは両方の木において、同一となり、採用されるルートにはいかなる軽微なバリエーションもなくなるので、長い台地は木間の小さい無関係な違いにより切断されることはない。

【０２２６】

この実施例へのわずかな変更は、辿りの時間よりむしろ各リンクの辿りのコストを第１の木に格納することである。それから、これらのコストは（新たにそれらを計算するよりむしろ）第２の木の計算において、用いられる。この技術は、（辿り時間の違いによるにせよ、それまでたどってきたルートの違いによるにせよ、はたまたそれ以外の理由によるにせよ、２本の木の間で変動するかもしれないいかなるコスト関数についても使用できるだろう。

【０２２７】

＜選択点の後のガイダンス＞

一旦、選択ルートがわかると、我々が第一のコスト関数に含みたいかもしれない、または含みたくないかもしれない一範囲の異なる基準について、選択ルート各々を評価できる。例えば、我々は天候のせいで、選択されている道が変わるのを望まないかもしれないが、異なる選択肢がどれくらい同等であるかに非常に興味を持っているかもしれない。特に１つのルートが乾燥していると予測され、別のものは濡れているかもしれない場合そうである。更なるルート計算なしで、我々は、選択ルートごとに、これらの他の基準を計算して表示し、それらのルートをそれらに応じた順序で（例えば最良の天気を最初に、またはカーボン排出量の低いものから、または停車箇所が良好に散らばっているものから）表示できる。

【０２２８】

一旦選択ルートがわかると、我々は一つ以上の選択肢が分岐するジャンクションも計算できる。先に説明したように、ジャンクションに十分先立って、我々はドライバにこれらの選択肢を、それらの特性に関する情報とともに提示できるので、ドライバは単に彼らがどちらの道を行くことを望んでいるかを選び、それに沿ってドライブすることができる。

【０２２９】

ドライバがルート・ガイダンスを要請した場合、どのようにそれが選択点で働くべきであろうか？道路標識としての選択肢の表示は、１つの出口が強調された通常のジャンクション・ダイヤグラムの良い置換物である。そして、音声ガイダンスは、ジャンクションに十分先立ってなされているとき、同様に２つ以上のオプションを提供できる。

【０２３０】

ドライバはここで、ボタン、タッチスクリーン等の何らかの従来のインタフェースによって、または、音声命令によって、ルートの１つを選択することにより、彼らが導かれることを望むルートを選ぶことができる。しかしながら、我々は、新規な方法を提供できる。我々はジャンクションが選択の行われることになっている場所であることは知っているので、我々はそこでのドライバのふるまいをルート間の正の選択として解釈できる。車両が選択ルートのうちの１つ上のジャンクションから離れる場合、我々はそれをドライバに

よる選択と解釈する。しかし、今度はどのようにして我々はガイダンスを実行するであろうか？我々は、単に選択ルート自体を用いることができ、各更なるジャンクションで、ドライバをそのルート上の次の道部分へ導くことができる。しかし、ドライバがそのルートから逸脱する場合、何が起こるか。

【０２３１】

従来のガイダンスシステムであれば、現在位置から目的地まで最適ルートを再計算して、それに沿ってガイダンスを再開するであろう。我々の問題は、この新たな最適ルートは、選ばれた選択ルートでないかもしれないので、ドライバは、彼ら自身が選んだ道に沿って案内されていないのに気づくだろうということである。一旦ドライバが、選ばれたルートに沿って十分遠くにきて台地上にいると、目的地までの最適ルートが選択ルート自体であることがわかる。しかし、ルートが台地部分に着くまでは、一般に、この台地を用いないより最適なルートがある。これは、我々に１つの可能な解法を与える。

【０２３２】

ドライバがジャンクションから離れたあと、我々はガイダンスを、選択されたルートのために台地上にあるポイントに切替え、その点が一旦通過されると、我々は目的地へのガイダンスを再開する。我々は台地上のいかなる点、例えば始部、中央または末端を用いることができた。しかし、この中間のガイダンス点ができるだけ早く落とされることができるようにするために、我々は、台地上の到達した第一の道部分を用いることを勧める。ドライバがその点を通過するとき、我々は目的地へガイダンスを切替える。いずれにしろそれは今や選択ルートに沿っているからである。第１の台地に到達する前に他の選択点があるという可能性もある。その場合、我々は、単に次の選択点をガイダンスのための一時的な目的地とし、一旦その点が通過されると、再びガイダンスを目的地に切り替える。

【０２３３】

ドライバが中間のガイダンス点を通過しないという可能性がまだある。そして、それは経由地点またはバイアが旅に加えられることが可能である従来のシステムが直面する問題である。したがって、従来の解法のいずれであっても用いることができる。

例えば、ドライバが、我々がそれらを導いているルートからそれて、目的地木からわかるように、バイアへの新しく計算された最適ルートが、ある距離（またはコスト）だけ最終目的地から離れるものになれば、我々は、あたかもそのバイアに達したかのようにそのバイアへのガイダンスをキャンセルし、その後に来る次の選択点、または、最高の良好性を有するルート上の台地の始まりのうちの最初に到達したものからガイダンスを続けることができる。

【０２３４】

これらの技術により、我々が提示したルート間でドライバがそれらのルートの１つに沿って行くことにより彼らの選択を行えるようにでき、しかも、彼らが導かれたルートからそれる場合であっても、必要であれば、我々はガイダンスをなおも提供できる。

【０２３５】

＜高速迂回路検索＞

一旦ソース木および目的地木が合成されると、我々は合成木において、いかなるノードを介してもソースから目的地へ行くことのコスト（または時間または距離）を計算したことになる。これはそれ自体において、興味がある特定の場所を伴うルートを見つけるために役立つことがありえる。というのは、どのようなノードを経たルートであっても、配線アルゴリズムを再度走らせる必要なく、上記２本の木において、たどることができるからである。

【０２３６】

我々が多数のノードのいずれかを介して行くコストを考慮したい場合、これは特に重要でありえる。これの例は、我々が旅で良い停止場所を探したいと思っている場合である。選ばれたルートに対して、我々は、直線距離にしてそこから所定の距離の範囲内に位置するすべての興味のある地点（おそらくレストラン、化粧用施設、ガソリンスタンド、休息所）を見つけることができる。我々はこれを付近検索と呼び、これを行うための周知の技

術がある。１つの技術は、最小限の垂直な距離を見つけるためにルート上の各リンクまでの垂直な距離を計算することであり、別のものは、ドライブ距離の下限を見つけるためにルート上の各ジャンクションまでの直線距離を計算することである。通常は、我々は、ルートの両側の５ｋｍのバンドの範囲内のすべてのレストランまたはルート上のジャンクション（いかなるものであっても）の１ｋｍの円内のすべてのガソリンスタンドを見つけるかもしれない。ここで、レストランまたはガソリンスタンドは直線距離にすれば近接しているけれども、それらは、大量の余分のドライブ（おそらく１０ｋｍ外に出て、次の高速道路ジャンクションからバック）を伴うことがありえる、または、それらは、ごくわずかな余分のドライブ（１つのジャンクションを離れ、次のものと平行して移動）しか伴わないことがありえる。

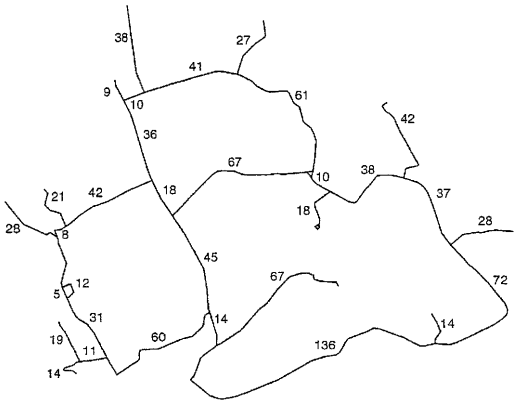
【０２３７】

余分のドライブ時間および距離は、新たなルート計算がそれらを見つけることを必要としない。我々が合成木を見る場合、我々はすでに、レストランがあるリンクを介してソースから目的地へ行くコストを持っている。したがって、我々は、そのリンクを経由するソースから目的地までの最適ルートの時間および距離を持っている。これは、一般に、近くの選択ルートのバイア・コスト（そのルートのための台地上のリンクを介して行くコストとして入手可能）よりも高く、一般に、その選択ルートからのわずかなそれ（軽微な迂回）に過ぎない。さらに重要なことに、そのレストランを経由することは、それが同じ道を往復しようが、出て行くときと戻るときとで道が異なっていようが、多くの場合選択ルートからの最小の迂回である。

【０２３８】

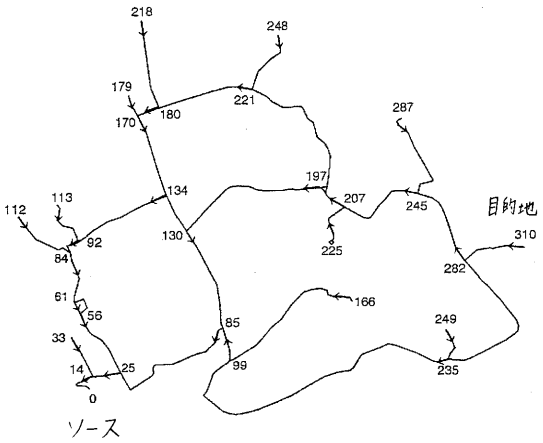
したがって、我々は、付近検索で見つかったレストランをフィルターに通すことができ、所定の閾値量よりも小さい量を我々の全体の旅コストに加えるものだけを保持できる。もちろん、我々は、木を造るために用いられた同じコスト関数を用いる必要はない。それらの木は、それらが最小にしているコスト関数に加えて、累積的な時間、距離、安全性または他の要因を含むことができ、これらを、しきい値処理において、用いることができる。例えば、我々は、我々の推定ドライブ時間に１０分以下しか加えないすべてレストランを探してもよい。合成木を用いずにこれを行うことは、付近検索によって、見つかったノードごとに二回ルーティング・アルゴリズムを走らせることを伴うだろう。そして、それは、コードを複雑化し、非常に長い時間がかかり、貴重なＣＰＵおよびメモリ資源を消費することになるだろう。我々の場合は、そうではなく、すでに上記ソース木および目的地木で格納され、合成木で合計された合成コスト、時間または距離を調べさえすればよい。我々がなおも現実的であると思うものに対して、我々は、ソース木および目的地木において、最も近いリンクからそのルートをたどり（トレースバックし）、高価なルーティング・アルゴリズムを走らせる必要なく、迂回ルートを得ることができる。そして、これらのルートを、道タイプ、渋滞、天気、その他のドライバが重要と考えることについて評価できる。

【図１】



距離

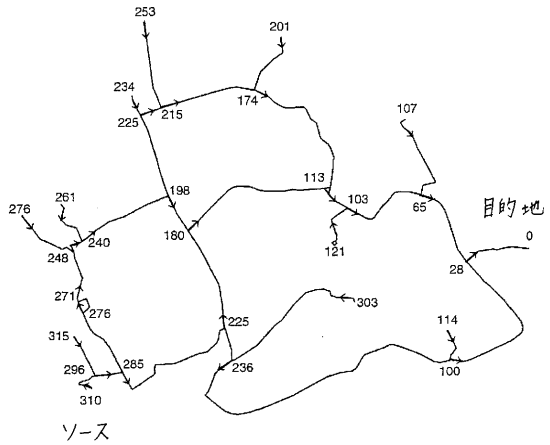
【図２】



ソース

ソース木

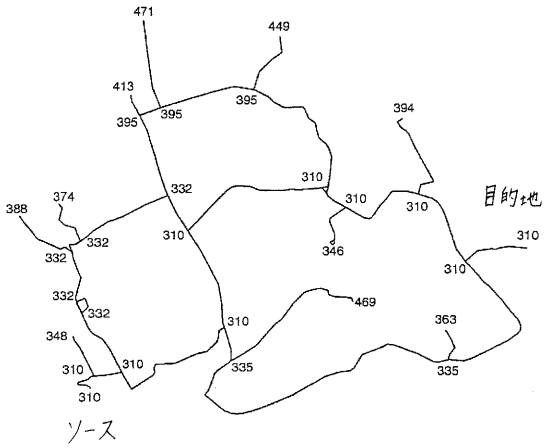
【図３】



ソース

目的地木

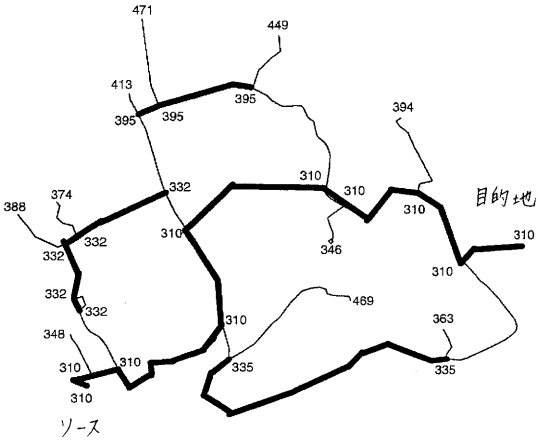
【図４】



ソース

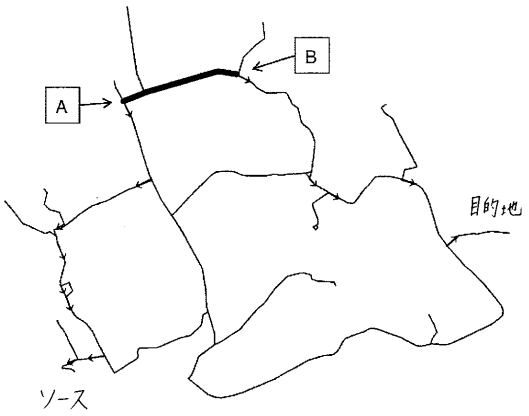
合成木

【図５】



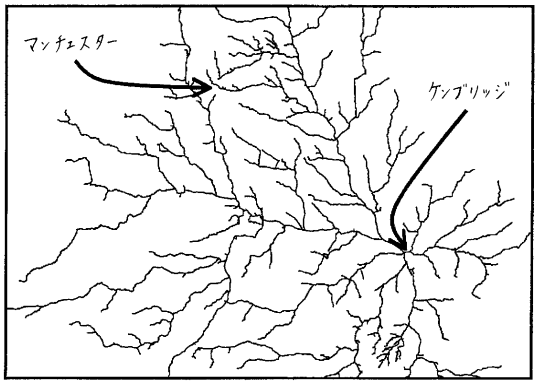
台 地

【図６】



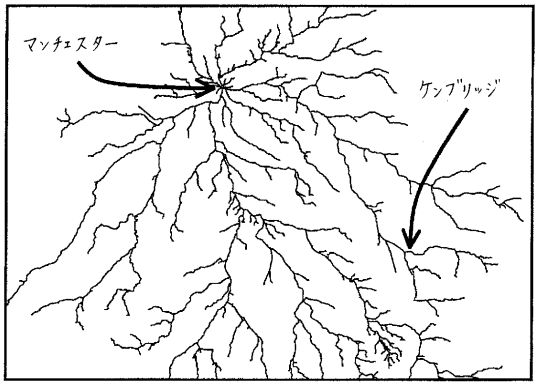
台地の良好性

【図７】



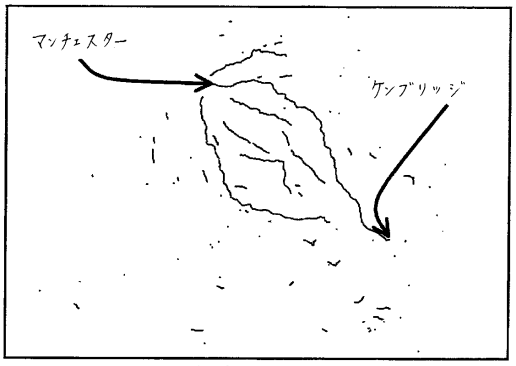
ケンブリッジからのソース木

【図８】



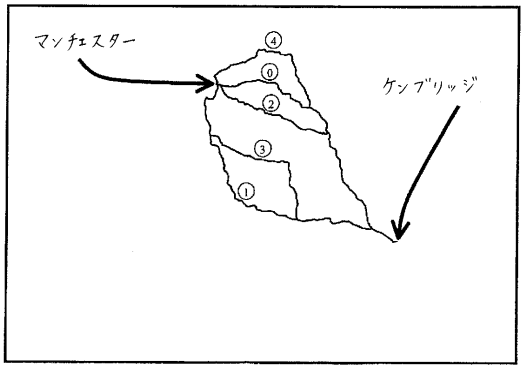
マンチェスターへの目的地木

【図９】



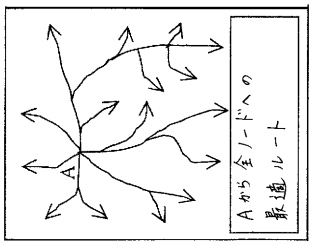
合成木

【図１０】

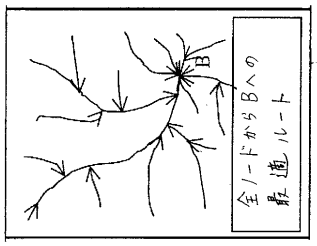


台地ルート

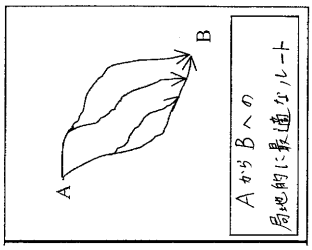
【図１１】



【図１２】



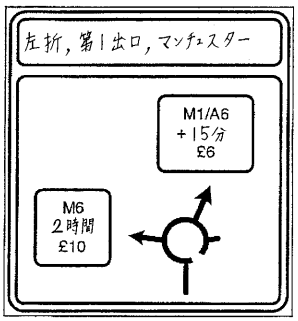
【図１３】



【図１４】

ソート：	精通度	高速道路	時間	距離	料金
A14/A1/A606/A6			3hrs 13min	154マイル	£4.00
A14/A1/A57/A628			3hrs 21min	162マイル	£4.00
A14/M1/A6			3hrs 20min	172マイル	£10.00
A14/M6 toll/M6			3hrs 01min	178マイル	£14.00
A14/M6			3hrs 06min	180マイル	£10.00

【図１５】



【図１６】

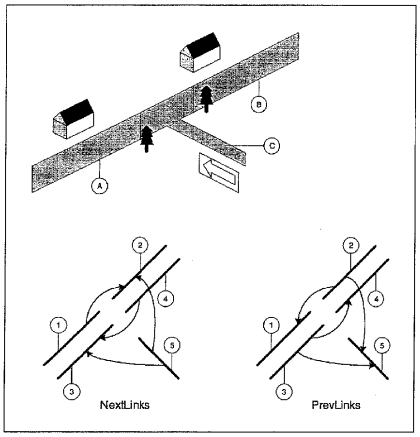
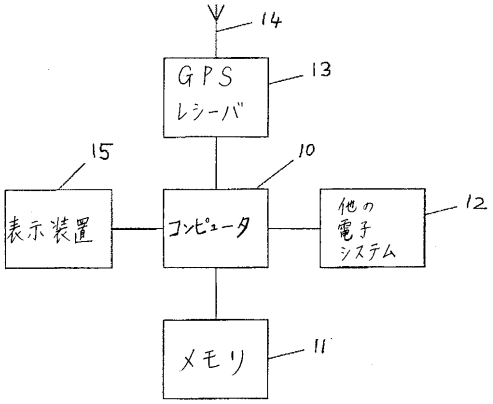
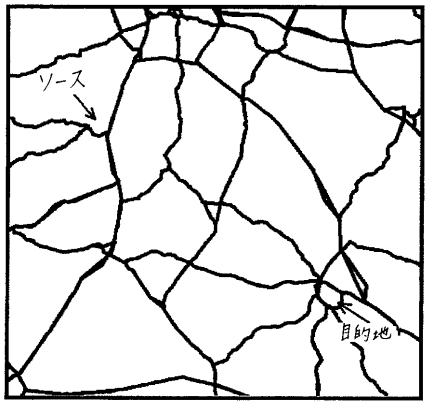


Fig 16

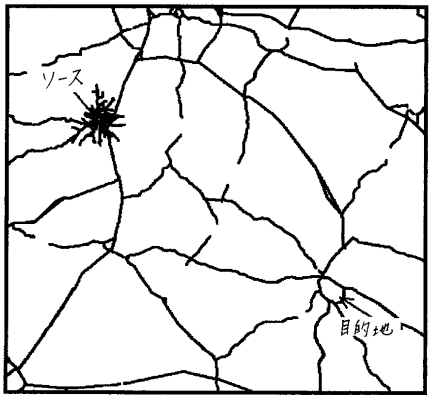
【図１７】



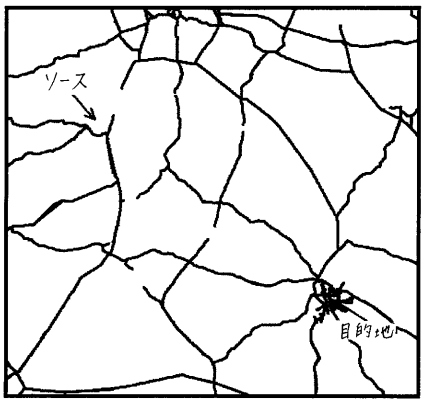
【図１８】



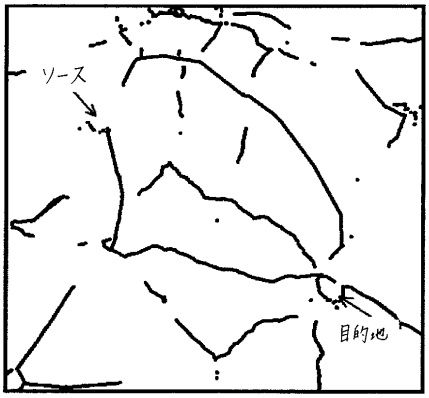
【図１９】



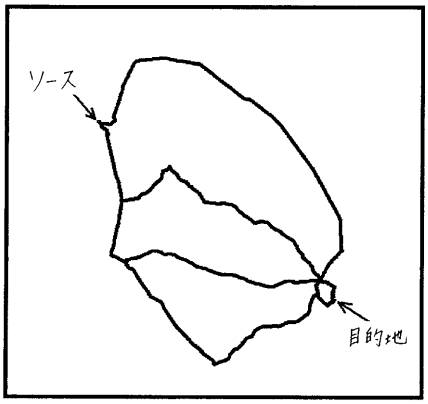
【図２０】



【図２１】



【図２２】



フロントページの続き

(72)発明者　アラン・ヘンリー・ジョーンズ
英国シービー２４・９ワイピー、ケンブリッジシャー、ケンブリッジ、ヒストン、ナロー・レイン
６７番

審査官　池田　貴俊

(56)参考文献　米国特許第０６４０１２３４（ＵＳ，Ｂ１）
米国特許第０６３７７５５１（ＵＳ，Ｂ１）
米国特許出願公開第２００４／００３９５２０（ＵＳ，Ａ１）
特開２００５－００９９７８（ＪＰ，Ａ）
特開２００４－００４１３１（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０１Ｃ　２１／３４