情報処理学会論文誌　プログラミング　Vol. 4　No. 2　116–133 (Mar. 2011)

領域限定言語に基づく最適経路問合せ

森畑明昌[†1]　松崎公紀[†2]　武市正人[†3]

グラフ中の最短経路を求める問題，またはより一般に何らかの意味で最適な経路を求める問題は，非常に多くの応用を持つ重要な問題である．そのため，最短経路問題やその変種はさかんに議論され，様々なアルゴリズムが提案されてきた．しかし，これらの成果はアルゴリズムになじみのない人には利用が難しい．自分の解きたい問題に応じて適切にアルゴリズムを選択するのは一般に難しく，またそれぞれのアルゴリズムに効率の良い実装を与えるのも容易でない．本論文では，領域限定言語に基づいた最適経路問合せ手法を提案する．提案手法では，発見したい最適経路の仕様を領域限定言語によって記述する．この言語は，その記述から最適経路問合せを行う効率の良いアルゴリズムが機械的に導出できるよう設計されている．また，この言語では最適経路問合せに関する既知の問題クラスの多くを自然に記述できる．よって，アルゴリズムに関する知識をまったく要することなく広い範囲の最短経路問合せを効率良く行うことができる．我々は実際に提案手法を実装した．このシステムは，最適経路の仕様記述をもとに，効率の良い最適経路問合せを行うプログラムを出力する．最短経路問題の実装に関する既知の成果を利用し，また最適経路問合せに特有のいくつかの効率化を加えることにより，我々のシステムによる最適経路問合せは，既存のライブラリに比べても，その利用が簡便であるだけでなく実行効率も良いことが確認された．

Optimal Path Querying Based on a Domain-specific Language

Akimasa Morihata,[†1] Kiminori Matsuzaki[†2] and Masato Takeichi[†3]

The problem to find the shortest path in a graph, or those to find the optimal path according to some criterion of optimality, are very important because of their numerous applications, and there are many studies on this topic. However, algorithmic results on solving optimal path problems are not very accessible for nonspecialists. It is difficult to choose an algorithm that enables us to achieve our objective, and moreover, its efficient implementation should be nontrivial. In this paper, we propose a method of optimal path querying. Our method is based on a domain specific language. The language is designed so that we can mechanically derive an efficient algorithm to find the described path. Therefore, we can efficiently find the optimal path with little algorithmic knowledge. Describing the specification of desirable paths by the language suffices. We implemented the method as an optimal path querying system. The system generates a program code for optimal path querying from the description of the optimal path. By virtue of a known efficient implementation of shortest path problems and our careful tuning, our system is not only easier to use but also more efficient than an existing library.

1. 序　　論

たとえば自動車で旅行に出ることを考える．このとき，目的地への経路が問題となることは少なくないだろう．普通は最速の経路が望まれるが，これを求めるのも簡単ではない．特に渋滞などの要素を加味すると難しい．また，あえて最速ではない経路を選ぶことも少なくない．たとえば，予算の観点から高速道路の必要以上の利用は避けたい，と思うのは自然である．ある時刻に間に合う範囲で最も安い経路を求めるのは，単に最速の経路を求めるのよりも難しい．また，長旅であれば，途中の景勝地などを訪れることも考えられる．いくつかの景勝地をめぐり，かつある程度早く到着したい，などと考えると問題はかなり複雑になる．

本論文では，グラフ中から何らかの意味で最適な経路を発見する問題，すなわち最適経路問題を考える．最適経路問題は最短経路問題の一般化にあたり，様々な応用を持つ．たとえば，先ほどあげたような旅行の計画を立てる問題などは典型的な応用の 1 つである[2),7)]．ほかにも，たとえば通信ネットワークの解析にも最適経路問題は用いられる[31)]．遅延などが一定の基準を満たす経路の発見が解析の基礎をなすからである．また，グラフ構造データベースへの問合せに最適経路問題が利用されることもある．正規表現経路問合せ[14),22),32),33)] はグラフ構造への問合せとして有用であるが，一般に 1 つの問合せにマッチする経路は無数にあるため，何からの意味で最適なものを代表として提示する必要がある．さらに，非常に広い範囲の組合せ最適化問題が最短経路問題に帰着できることも知られている[4),26)]．その

†1 東北大学電気通信研究所
Research Institute of Electrical Communications, Tohoku University
†2 高知工科大学情報学群
School of Information, Kochi University of Technology
†3 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
この研究の初期の成果は文献 35) として報告した．

ため，最適経路問題は最短経路問題に帰着できる様々な問題およびその変種を解く強力な手法となりうる．

このように，最適経路問題は理論実用の両面から非常に重要であり，様々な問題クラスとそれを解くための効率の良いアルゴリズムが考えられてきた．2 章ではこれらの成果の一部を概観する．しかし，これらは専門家以外には利用しがたい．まず，多くの問題クラスと多くのアルゴリズムがあるため，自分の解きたい問題にとって適切なものを発見するのは容易でない．さらに，提案されているアルゴリズムに対し，実際に効率の良い実装を与えるのも簡単ではない．また，考えている問題は効率の良い解法を持たない，または効率の良い解法が知られていない問題である可能性もある．この場合，既知の効率の良いアルゴリズムで解けるよう，目的を適切に定式化しなおす必要がある．

我々の目的は，アルゴリズムに関する知識がほとんどない人が最適経路問題に関する成果を利用できるようにすることにある．この目的を達成するため，我々は最適経路問合せのための領域限定言語を設計した．この言語は発見したい最適経路の仕様を記述するためのものであり，最適経路問題に関する既知の問題クラスの多くが自然に記述できるように，そして同時に，この言語による記述をもとに効率の良い最適経路問合せアルゴリズムが機械的に導出できるように，設計されている．よって，我々は最適経路の仕様を書き下すだけで，広い範囲の最適経路問題を効率良く解くことができる．また，この言語は効率的な最適経路問合せに様々な問題を帰着する際の指針も与えることになる．

提案した言語に基づき，我々は実際に最適経路問合せシステムを試作した．提案手法では最短経路アルゴリズムに似たアルゴリズムを用いる．そのため，我々は最短経路アルゴリズムの効率に関する調査[10),11),38)]をふまえ，さらに最適経路問合せ特有のいくつかの点を考慮し，効率の良い実装を与えた．

我々の主な貢献は以下の 3 点である．

- **最適経路問合せのための領域限定言語の提案**．我々は最適経路の仕様を記述するための領域限定言語を設計した．この言語では最適経路問題に関する既知の問題クラスの多くを自然に記述できる．
- **最適経路問合せアルゴリズムの導出**．我々は，提案した言語による最適経路問題の記述から最適経路を発見するアルゴリズムを機械的に導出する方法を示した．これは最適経路問題を最短経路問題に帰着することによって達成される．
- **最適経路と問合せシステムの実装**．我々は，提案手法を最適経路問合せシステムとして実装した．同時に，効率の良い実装のためのいくつかの点について議論を行った．この結果，我々のシステムは，C++でのグラフアルゴリズムの標準的な実装である C++ Boost Graph Library[44)] のものに比べても高速な最適経路問合せを達成できた．

本論文の構成は以下のとおりである．まず，2 章では既存の最適経路問題に関する研究成果を概観する．次に，3 章では，今回提案する最適経路問合せのための領域限定言語を示し，その表現力について議論する．4 章では，領域限定言語による最適経路の記述から最適経路問合せアルゴリズムを導出する手法を示す．また既知の最適経路問合せアルゴリズムとの関係についても議論する．5 章では最適経路問合せシステムの実装方針と効率化のための工夫を述べる．6 章では実装した最適経路システムの効率を既存のライブラリや人手によるプログラムと比較する．7 章では関連する研究について述べる．8 章では本論文をまとめ，さらなる発展の可能性について議論する．

本論文では，読者は最短経路アルゴリズムおよび形式言語理論について基本的な知識を持っているものと仮定する．必要ならば標準的な教科書，たとえば最短経路アルゴリズムについては文献 13)，形式言語理論については文献 24) などを参照されたい．

2. 最適経路問題

本章ではいくつかの最適経路問題およびそれを解くアルゴリズムについて概観する．なお，ここで述べるものは既知の最適経路問題の一部でしかない．たとえば，Dreyfus[19)] や Deo ら[16)] によるサーベイはさらに多くの研究を紹介している．

以下，特に断らない限り，頂点数 V，辺数 E，非負整数辺重みの有向グラフを考える．

2.1 最短経路問題

最も単純な最適経路問題の 1 つは 2 点間最短経路問題である．2 点間最短経路問題は，ある 2 頂点をそれぞれ始点および終点とする経路の中で経路中の辺の重み和が最も小さいものを求める問題である．2 点間最短経路を求めるアルゴリズムとしてはダイクストラ法[18)] がよく知られている．フィボナッチヒープ[23)] や relaxed ヒープ[20)] を用いることにより，ダイクストラ法は $O(V \log V + E)$ 時間で実行することができる．

2.2 制約最短経路問題

Joksch[28)] は，各辺が 2 種類の重みを持つグラフ上のある 2 点間を結ぶ経路で，片方の重みの和が一定値以下で他方の重み和が最小のものを求める問題を考え，これを制約最短経路問題と呼んだ．例としては，ある時刻に間に合うものの中で最も費用の安い経路を求める問題などがあげられる．彼は，この問題が NP 困難であることを示し，また動的計画法に基づく偽多項式時間のアルゴリズムを示した．このアルゴリズムは，各頂点について，

制約となる重みの値ごとに最良の経路を考えるものである．

2.3　時間窓制約最短経路問題

時間窓制約最短経路問題[17]は，各辺が経過時間とコストを持ち，各頂点はそれぞれ特定の時間帯にしか利用できない，というグラフの上で，2 点間を結ぶ最小コストの経路を発見する問題である．この問題は Joksch の制約最短経路問題の一般化である．制約最短経路問題ではすべての頂点が最初から利用でき同時に利用できなくなる．

Desrochers ら[17]はこの問題に対しダイクストラ法に似たアルゴリズムを与えた．彼らのアルゴリズムは，その頂点までの経路の中で他に経過時間とコストの両方で下回る経路が存在しないものを記憶し，記憶されている経路の延長を経過時間の短いものから順に吟味してゆくことで最適経路を得る．彼らは，アルゴリズム中で用いるデータ構造ごとの計算量を示している．どの構造を用いるべきなのかはグラフの性質による．たとえば，各頂点の時間窓の幅がそれほど変わらず，また $E \leq N^2$ を満たすグラフ上では，時間窓の幅の最大値を k として $\mathrm{O}(kE)$ 時間で最適経路を求めることができる．なお，彼らは各辺の経過時間は正であることを要求する一方，コストは任意の実数値を許している．

2.4　資源制約最短経路問題

時間窓制約最短経路問題をさらに一般化した問題も複数考えられている．Irnich ら[27]はこれらを資源制約最短経路問題と呼んでいる．彼らは，既存の様々な資源制約最短経路問題を概観し，それらに対するアルゴリズムが，各頂点への特定の条件を満たす経路を記憶しそれらを延長しつつ最適経路を発見する，という共通の構造を持つことを指摘している．

2.5　経由頂点指定最短経路問題

経由頂点指定最短経路問題[19),25)]は特定の頂点集合を経由してある 2 点間を結ぶ経路の中で最短のものを求める問題である．これは買い物の経路設定などで日常的に現れる．また，巡回セールスマン問題も経由頂点指定最短経路問題の一例である．

Dreyfus[19]は最短経路問題についてのサーベイの中でこの問題に言及している．彼のアルゴリズムは，k 頂点が経由地として指定された場合，まず始点終点を含む $k+2$ 点間の最短経路をそれぞれ計算し，次にこの $k+2$ 頂点のみがそれぞれ最短経路で連結されたグラフを考え，このグラフの上で始点終点が指定された巡回セールスマン問題を解くものである．$k+2$ 点間の最短経路の計算にダイクストラ法を用いた場合，$\mathrm{O}(k(V \log V + E) + 2^k)$ 時間でこの問題を解くことができる．

Ibaraki[25]は Dreyfus のものとは異なるアルゴリズムを示している．これは，もとのグラフより大きなグラフ上での最短経路問題へと問題を帰着させるものであり，本質的には次に述べる Romeuf の手法のインスタンスとなっている．帰着先の問題をダイクストラ法で解くとすると，この解法の時間計算量は $\mathrm{O}(2^k V(k + \log V) + 2^k E)$ となる．

2.6　正規言語制約最短経路問題

正規言語制約問題[39]は，ある 2 点間を結ぶ経路の中で，それを文字列と見なしたときにある正規言語 $\mathcal{L}$ に含まれる最短のものを見つける問題である．たとえば，経由頂点指定最短経路問題，ある特定の経路集合を用いない経路の中で最短のものを求める問題[45]，白黒 2 色の辺があるグラフの上で白の辺と黒の辺を交互にたどって 2 点間を結ぶ経路を発見する問題，などが正規言語制約問題の具体例である．

Romeuf[39]は正規言語制約最短経路問題が最短経路問題に帰着できることを示した．着想は辺集合をアルファベットとする有限状態オートマトンを考えることである．このとき，グラフは始点と終点を結ぶ経路を認識する有限状態オートマトンと見なすことができ，最短経路はそのオートマトンの受理する語のうちで最小重みを持つものに対応する．$\mathcal{L}$ に含まれる経路は $\mathcal{L}$ を認識する有限状態オートマトンとグラフの積オートマトンで認識できる．よって，この積オートマトンが受理する最小重み語を求めればよい．このアルゴリズムの計算量は，$\mathcal{L}$ を認識するオートマトンの状態数を k として，$\mathrm{O}(kV \log(kV) + k^2 E)$ である．

2.7　文脈自由言語制約最短経路問題

Barret ら[3]は正規言語制約最短経路問題を一般化した形式言語制約最短経路問題について議論した．たとえば，文脈自由言語制約最短経路問題は正規言語制約最短経路問題と同様に定義できるが，これに対して彼らは $\mathrm{O}(NV^2 \log(NV) + RV^3)$ 時間の動的計画法アルゴリズムを与えている．ここで N と R はそれぞれ制約をチョムスキー標準型の文脈自由文法で表した際の非終端記号と生成規則の数である．このアルゴリズムは，各 2 頂点を結び各非終端記号から生成されうる最短の経路を順次求めてゆくものである．

2.8　時間依存最短経路問題

これまであげた問題はある一定の制約の下で最短経路を求める問題であった．Cooke ら[12]は状況によって辺重みが変化するグラフを考えることは応用上重要であることを指摘した．たとえば，時刻表をもとに鉄道の最適な乗換を求める場合や，日中は渋滞しがちな道路を扱う場合などがこの範疇に入る．彼らは離散的な時刻によって辺重みが変動するグラフ上での最短経路を求める動的計画法アルゴリズムを示した．これは，各時刻にその頂点に到達する経路の中で最小重みのものを求めるものである．

Orda ら[36]はこれをさらに発展させ，辺重みが実数上の任意の関数の場合など，より一般的な状況で効率の良いアルゴリズムが存在するための十分条件を議論した．時間依存最短

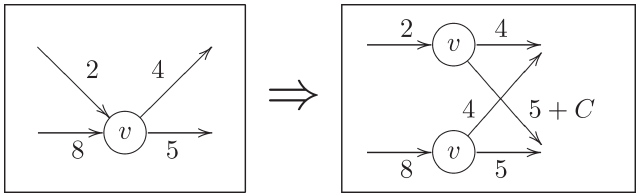

図 1　方向依存最短経路問題のための頂点および辺の複製．重み 2 の辺を渡って頂点 v にたどり着いた場合には重み 5 の辺の重みが C 増加する状況を表している
Fig. 1　A graph replication for the approach-depenedent shortest path problem: it describes the situation where the edge weight 5 becomes $5 + C$ when we come from the edge of weight 2.

経路問題は，重みが FIFO（First-In First-Out）条件を満たせば，すなわち早く辺を渡り始めた方が早く辺の終端にたどり着く，という性質が成り立てば，ダイクストラ法と同様にして解くことができる．たとえば，渋滞する道路は FIFO を満たすが，急行列車が普通列車を追い抜く鉄道は FIFO を満たさない．彼らは，いくつかの状況下，たとえば各頂点で任意時間の待機が許可されている場合など，では問題を FIFO を満たすグラフ上の問題に帰着できることを示した．たとえば，先にあげた鉄道の例であれば，後発の急行を待つという行為に対応する辺を考えることで FIFO だと見なすことができる．

2.9　方向依存最短経路問題

方向依存最短経路問題[8] も古くから議論されてきた最短経路問題の変種である．これは，辺重みが直前に渡った辺の始点に依存する最短経路問題であり，たとえば，鉄道経路探索での乗換の待ち時間などを表現することができる．Caldwell[8] はこの問題がより大きなグラフの最適経路問題に帰着できることを示した．具体的には，直前にいた頂点ごとにそれぞれの頂点を複製する．この様子を**図 1** に示す．今，グラフが左のような形状をしており，重み 2 の辺を渡って頂点 v にたどり着いた場合には重み 5 の辺の重みが C 増加するとする．この場合，直前にいた頂点ごとに v および v を始点とする辺を複製し，図 1 右のグラフを作る．上方の v は重さ 2 の辺を渡ってきた状況に対応し，よって辺重みが $5 + C$ となる．これにより辺重みの方向依存性が除去されるため最短経路は簡単に求めることができる．ダイクストラ法を用いるとして，時間計算量は $\mathrm{O}(V^2 \log V + VE)$ となる．

2.10　最大容量路問題

多くの研究が辺重みかそれに類するものの和を最小化する問題を考えている．これに対し，最大容量路問題では，ある 2 点間の経路の中で，その経路に含まれる辺の容量の最小値が最大となるものを求める．Carré[9] はこの問題を含むいくつかの問題が最短経路問題と同様のアルゴリズムで解けることを述べた．また，Punnen[37] は，中央値を線形時間で求めるアルゴリズムを用い，無向グラフに対して最大容量路を $\mathrm{O}(V)$ 時間で求める分割統治アルゴリズムを与えた．このアルゴリズムは辺集合を重みの大小によって再帰的に 2 等分する．重みの大きな辺のみからなる経路が存在すればそれが最適解である．またそのような解が存在しない場合，重みの大きな辺をすべて縮約した小さなグラフを考えれば十分である．

2.11　方向依存・時間依存・正規言語制約最短経路問題

今まであげたようないくつかの要素の複合も議論されている．たとえば，Brodal ら[7] は辺重みが時間に依存するグラフ上での正規言語制約最短経路問題を考えている．Sherali ら[43] はこれをさらに発展させ，方向依存・時間依存・正規言語制約最短経路問題に対するアルゴリズムを与えている．また，資源制約最短経路問題の場合と同様，各頂点に対し複数の経路を管理することで，このアルゴリズムを効率良く実現できることも述べている．

2.12　k 番目最短経路問題

短い方から k 通りの経路を発見する問題は k 番目最短経路問題と呼ばれ，やはり古くから多くの研究がある．この問題に対しては Eppstein[21] による高速なアルゴリズムが知られている．この手法では，始点と終点を結ぶ経路を短い方から順に列挙するヒープ構造を構成し，ヒープ中の要素の上位 k 件を取り出す．時間計算量は $\mathrm{O}(V \log V + E + k)$ である．

2.13　全点間最適経路問題

これまでの例ではある 2 点間の最適経路を求める問題を考えてきた．多くの場合，似た手法によってすべての 2 点間の最適経路を求めることもできる．たとえば，Flesca ら[22] は，グラフ構造データベースに対する正規言語制約全点間最短経路による問合せが有用であることを指摘した．また，辺重みが非負であるという仮定なしに，正規言語制約に対応する有限状態オートマトンの状態数を k として，この問合せが $\mathrm{O}(k^3 V^3)$ 時間で達成できることも述べている．アイデアは Romeuf のものと非常に近く，制約に対応する有限状態オートマトンとグラフとの積オートマトンに対応するグラフ上で全点間最短経路問題を解くものである．

また，単純に 2 点間最適経路を求めるアルゴリズムを複数回実行することで全点間最適経路を求めても，多くの場合にはそれほど遅くはない．たとえば，ダイクストラ法が 1 頂点から全頂点までの最短経路を一度に求められることに注意すると，辺重みが非負の場合には，Romeuf のアルゴリズムを V 回用いることで正規言語制約全点間最短経路を $\mathrm{O}(kV^2 \log(kV) + k^2 VE)$ 時間で求めることができる．

3. 最適経路問合せのための領域限定言語

以下，有向グラフ G を考える．グラフ G は頂点集合 V と辺集合 E からなる．誤解のない範囲で，これらはその要素数の表現にも用いる．v に添字がついたものは頂点，e に添字がついたものは辺を表す．辺の列 $e_1, \ldots, e_n$ が経路であるとは，すべての辺 e_i（$1 \leq i < n$）について，e_i の終点と e_{i+1} の始点が一致することである．頂点を 0 辺からなる経路と見なすこともある．経路は x や y などを用いて表す．経路の最初の辺の始点を経路の始点，最後の辺の終点を経路の終点と呼ぶ．$x \overset{e}{\rightsquigarrow} y$ は x の終点と y の始点を辺 e でつないで得られる経路を表す．グラフに関する値は，頂点または辺から値を計算する基本関数によって表現する．たとえば辺重みを与える関数などが基本関数である．

3.1 構 文 文 法

提案手法では，**図 2** の言語を用いて最適経路問合せの仕様を記述する．プログラムのトップレベルは 1 つの非負整数値関数名と 1 つの真偽値式からなる．前者を目的関数と呼び，後者を制約条件と呼ぶ．目的関数と制約条件はそれぞれ経路に対する再帰関数で定義されている．再帰関数の定義にはそれぞれの型ごとに標準的な演算および基本関数を用いることができる．基本関数は真偽値または非負整数値を経路によらず返すのであれば任意のものでよい．整数値式では減算および非負値を用いることができないこと，真偽値式では比較のオペランドの一方が非負整数でなければならないこと，に特に注意せよ．

$$
\begin{array}{lll}
\mathit{Prog} & ::= \mathsf{minimize}\ f_{int}(x)\ \mathsf{s.t.}\ \mathit{Exp}_{bool}\ \mathsf{where}\ \mathit{Decl} \cdots \mathit{Decl} & \{\text{プログラム}\} \\
\mathit{Decl} & ::= \mathsf{int}\ f_{int}(v) = \mathit{Exp}_{int};\quad f_{int}(x \overset{e}{\rightsquigarrow} v) = \mathit{Exp}_{int}; & \{\text{非負整数値関数}\} \\
 & \quad | \quad \mathsf{bool}\ f_{bool}(v) = \mathit{Exp}_{bool};\quad f_{bool}(x \overset{e}{\rightsquigarrow} v) = \mathit{Exp}_{bool}; & \{\text{真偽値関数}\} \\
\mathit{Exp}_{int} & ::= \langle \text{unsigned\_integer} \rangle \mid \mathit{Exp}_{int} + \mathit{Exp}_{int} \mid \mathit{Exp}_{int} \times \mathit{Exp}_{int} & \{\text{非負整数値式}\} \\
 & \quad | \quad p_{int}(v, e) \mid f_{int}(x) \mid \mathsf{if}\ \mathit{Exp}_{bool}\ \mathsf{then}\ \mathit{Exp}_{int}\ \mathsf{else}\ \mathit{Exp}_{int} & \\
 & \quad | \quad \max(\mathit{Exp}_{int}, \mathit{Exp}_{int}) \mid \min(\mathit{Exp}_{int}, \mathit{Exp}_{int}) & \\
\mathit{Exp}_{bool} & ::= \mathit{true} \mid \mathit{false} \mid \mathit{Exp}_{int} \preceq \langle \text{unsigned\_integer} \rangle \mid p_{bool}(v, e) & \{\text{真偽値式}\} \\
 & \quad | \quad f_{bool}(x) \mid \mathit{Exp}_{bool} \wedge \mathit{Exp}_{bool} \mid \mathit{Exp}_{bool} \vee \mathit{Exp}_{bool} \mid \neg \mathit{Exp}_{bool} & \\
\preceq & ::= \leq \mid \geq \mid < \mid > \mid = \mid \neq & \{\text{比較演算子}\}
\end{array}
$$

図 2 最適経路問合せの記述のための領域限定言語．x は経路変数，v は頂点変数，e は辺変数，f_{int} は整数値関数名，f_{bool} は真偽値関数名，$p_{int}(e, v)$ は非負整数値の基本関数，$p_{int}(e, v)$ は真偽値の基本関数

Fig. 2 A domain-specific language for optimal path querying: x, v, f_{int}, f_{bool}, $p_{int}(v, e)$, $p_{bool}(v, e)$ respectively denote a path variable, a vertex variable, an edge variable, a nonnegative-integer-valued function, a Boolean-valued function, a nonnegative-integer-valued primitive function, and a Boolean-valued primitive function.

この構文文法に加え，目的関数には付加的な制約がある．目的関数の定義の右辺式は，他の再帰関数の値を直接含んではならない．ここで，ある式がある関数の値を直接含むとは，その式の if 式の条件部でない部分式がその関数の呼び出しであることである．たとえば，式 $\mathsf{if} \cdots g(x) < n \cdots \mathsf{then} \cdots f(x) \cdots \mathsf{else} \cdots$ は f の値を直接含むが g の値は直接は含まない．

3.2 プログラムの意味

式および再帰関数の意味は自明に定義する．式は経路から値を計算する関数と見なし，経路 x に対する式 exp の値を $\mathit{exp}(x)$ と記述する．プログラムの意味は次のとおり定義する．

定義 1　グラフ G に対し，プログラム $P = \mathsf{minimize}\ f(x)\ \mathsf{s.t.}\ \mathit{exp}\ \mathsf{where} \cdots$ の意味 $[\![P]\!]_G$ は経路の集合であり，G 上の経路すべての集合を X_G として，

$$[\![P]\!]_G = \{\, x \mid x \in X_G \wedge \mathit{exp}(x) \wedge (\forall y \in X_G.\, \mathit{exp}(y) \Rightarrow f(x) \leq f(y)) \,\}$$

で定義される．G に対する P の評価とは $x \in [\![P]\!]_G$ なる経路 x を求めることである．

3.3 記　述　例

3.3.1 2 点間最短経路

2 点間最短経路はこの言語で簡単に記述できる．

$$
\begin{array}{l}
\mathsf{minimize}\ \mathit{cost}(x)\ \mathsf{s.t.}\ \mathit{from}(x) \wedge \mathit{to}(x)\ \mathsf{where} \\
\quad \mathsf{int}\ \mathit{cost}(v) = 0; \\
\qquad \mathit{cost}(x \overset{e}{\rightsquigarrow} v) = \mathit{cost}(x) + w(e); \\
\quad \mathsf{bool}\ \mathit{from}(v) = p_{\mathit{from}}(v); \\
\qquad \mathit{from}(x \overset{e}{\rightsquigarrow} v) = \mathit{from}(x); \\
\quad \mathsf{bool}\ \mathit{to}(v) = p_{\mathit{to}}(v); \\
\qquad \mathit{to}(x \overset{e}{\rightsquigarrow} v) = p_{\mathit{to}}(v);
\end{array}
$$

目的は非負整数値関数 cost の値を最小にする経路を発見することである．cost の値は基本関数 w によって与えられる辺重みの和である．始点と終点がそれぞれ指定された頂点であることが制約条件であり，真偽値関数 from および to がこれを表現する．これらの関数は，引数として与えられた頂点が指定された始点であるかどうか調べる基本関数 p_{from} および終点であるかどうかを調べる基本関数 p_{to} を用いて定義されている．

このプログラムはその他の問題を記述する際の雛形となる．以下，cost，from，および

to は上記のものとする．

3.3.2　3 点間最短経路

2 点間を結び，途中で特定の頂点を通る最短の経路を求める問題を考える．これは経由頂点指定最短経路問題であり，以下のプログラムで表現できる．

$$\begin{array}{l}\mathsf{minimize}\ cost(x)\ \mathsf{s.t.}\ from(x) \wedge to(x) \wedge via(x)\ \mathsf{where}\\ \quad \mathsf{bool}\ via(v) = p_{via}(v);\\ \qquad via(x \overset{e}{\rightsquigarrow} v) = via(x) \vee p_{via}(v);\end{array}$$

ここで p_{via} は引数が指定された経由点であるかどうかを調べる基本関数である．

3.3.3　乗換コスト付き最短経路

各辺が鉄道での移動に対応するものと徒歩での移動に対応するものに分かれており，鉄道への乗車の際には待ち時間として C 要する場合を考える．これは典型的な方向依存最短経路問題であり，以下のプログラムで表現できる．

$$\begin{array}{l}\mathsf{minimize}\ cost'(x)\ \mathsf{s.t.}\ from(x) \wedge to(x)\ \mathsf{where}\\ \quad \mathsf{int}\ cost'(v) = 0;\\ \qquad cost'(x \overset{e}{\rightsquigarrow} v) = cost'(x) + w(e) + \mathsf{if}\ walk(x) \wedge p_{train}(e)\ \mathsf{then}\ C\ \mathsf{else}\ 0;\\ \quad \mathsf{bool}\ walk(v) = true;\\ \qquad walk(x \overset{e}{\rightsquigarrow} v) = \neg p_{train}(e);\end{array}$$

真偽値関数 $walk$ は現在電車に乗っているかを基本関数 p_{train} を用いて調べる．これを用い，非負整数値関数 $cost'$ は，乗換のコスト C を加算しながら所要時間を計算する．目的は所要時間が最小となる経路の発見である．

3.3.4　乗換回数制約最短経路

鉄道の乗換検索では，乗換回数の比較的小さな経路の発見も重要である．以下のプログラムは乗換回数が K 回未満のものの中で最短の経路を発見する．

$$\begin{array}{l}\mathsf{minimize}\ cost(x)\ \mathsf{s.t.}\ from(x) \wedge to(x) \wedge transit(x) < K\ \mathsf{where}\\ \quad \mathsf{int}\ transit(v) = 0;\\ \qquad transit(x \overset{e}{\rightsquigarrow} v) = transit(x) + \mathsf{if}\ walk(x) \wedge p_{train}(e)\ \mathsf{then}\ 1\ \mathsf{else}\ 0;\end{array}$$

乗換コスト付き最短経路の場合と同様にして，関数 $transit$ は乗換の回数を数える．これが K 未満であることが要求されている．$walk$ は乗換コスト付き最短経路問題で定義されたものである．なお，実際のプログラムでは K は具体的な定数でなければならない．

3.3.5　部分和問題

様々な組合せ最適化問題は最適経路問題に帰着できる[4),26)]．ここでは一例として部分和問題を考える．これは，整数の集合が与えられたとき，その部分集合で和が特定の値に一致するものがあるかを調べる問題である．ここでは特に，正整数の集合に対して，特定の値 W に和が一致する最小要素数の部分集合を求めることを考える．

この問題は最適経路問題に帰着できる．まずグラフを用意する．各頂点はそれぞれの整数に対応し，一列に並んでいる．各頂点からは「その整数を選ぶ」「その整数を選ばない」のそれぞれに対応する 2 辺が次の頂点へ向けて出ており，前者の重みはその整数の値，後者の重みは 0 である．このとき，このグラフ中の経路で，重み和が W に一致し，重み 0 でない辺の数が最も小さいものを求めることで，部分和問題の解は得られる．

$$\begin{array}{l}\mathsf{minimize}\ count(x)\ \mathsf{s.t.}\ sum(x) = W\ \mathsf{where}\\ \quad \mathsf{int}\ count(v) = 0;\\ \qquad count(x \overset{e}{\rightsquigarrow} v) = count(x) + \mathsf{if}\ w(e) > 0\ \mathsf{then}\ 1\ \mathsf{else}\ 0;\\ \quad \mathsf{int}\ sum(v) = 0;\\ \qquad sum(x \overset{e}{\rightsquigarrow} v) = sum(x) + w(e);\end{array}$$

ここで w は辺の重みを表す基本関数である．

3.4　記　述　力

我々の領域限定言語は，最適経路問題に関する既知の問題クラスの多くを記述できるように設計されている．

2 点間最短経路問題は明らかに記述できる．

最大容量路問題は直接は記述できない．なぜなら提案言語は最大化問題を記述できないからである．各辺の重みをその順序がちょうど逆転するように付け替え，辺重みの最大値が最小となる経路を求めることで最大容量路を求めることはできる．

制約最短経路問題の記述は容易である．時間窓制約最短経路問題の場合，頂点数によらずたかだか定数種類の時間窓のみが課されているならば記述できる．

方向依存最短経路問題は簡単に扱うことができる．

経由地点指定最短経路問題の記述は容易である．より一般に，正規言語制約最短経路問題の記述も可能である．正規言語に対応する決定的有限状態オートマトンを再帰関数を用いて表現すればよい．たとえば，n 状態の決定的有限状態オートマトンは $\lceil \log_2 n \rceil$ 個の真偽値

関数や 1 から n までの値のみを出力する非負整数値関数で表現できる．しかし，一般には文脈自由言語制約最短経路問題はこの言語では記述できない．

時間依存最短経路問題も一般には表現できない．特に Orda ら[36] が考えたような実数値を扱う問題は扱えない．しかし，いくつかの典型的な例を記述することはできる．たとえば，目的関数中に max を用いることで，「特定時刻までの待機」を表現することはできる．特定時刻までの待機を含む問題は FIFO 条件を満たす時間依存最短経路問題の典型であり実用上有用である．また，Cooke ら[12] のような，特定の時間に辺重みが変化する問題は扱うことができる．

記述できる問題の組合せにあたるような問題，たとえば方向依存正規言語制約最短経路問題，などは簡単に記述できる．これは領域限定言語の強みである．

我々の言語は 2 点間最短経路問題の記述に特化したものではない．たとえば，先にあげた 2 点間最短経路問題を例にあげると，基本関数 p_{from} や p_{to} が特定の頂点に対してのみ真となる必要はない．しかし，我々の言語は全点間最適経路を求めるようには設計されてはいない．全点間最適経路の計算はすべての 2 頂点対について繰り返し問合せを行うことで達成できる．

また，我々の言語では k 番目最短経路問題を記述することもできない．k 回にわたって「今まで発見されたものではないもの」を問い合わせることでこの解を求めることはできる．

4. 最適経路問合せアルゴリズム

4.1 グラフ拡大に基づく最適経路アルゴリズム

提案言語は，その言語による問題の記述から効率の良い問合せアルゴリズムが自動的に導出できるよう設計されている．

基盤となっているアイデアは Romeuf の正規言語制約最短経路問題の解法や Caldwell による方向依存最短経路問題の解法に近い．経路 $x \overset{e}{\rightsquigarrow} v$ が頂点 v までの最適経路であるならば x は e の始点までの最適経路である，という性質（以下 Bellman[4] にならいこの性質を「最適性の原理」と呼ぶ）が成り立てば，ダイクストラ法のように各頂点までの最適経路を順に求めてゆくことで最適解を得ることができる．しかし，最適経路問題の場合には最適性の原理は一般には成り立たたない．そのため，Romeuf や Caldwell の手法のように，最適性の原理が成り立つようにグラフを拡大する必要がある．以下，いくつかの例について，どのようにグラフを拡大すれば最適性の原理が成り立つようになるのかを，なぜ最適性の原理が成り立たないのかに着目しつつ考えてみる．

方向依存最短経路問題であれば，多少遠回りをして方向転換によるペナルティを避ける方が最終的なコストが小さくなる可能性がある．例として図 1 を再考する．重み 2 の辺と重み 8 の辺は始点から等距離の頂点に，重み 5 の辺は終点につながっているとし，また $C > 6$ とする．この場合，図中の頂点へは重み 2 の辺が最短経路を与えるが，終点までなら重み 8 の辺を利用する方がコストが小さい．このように，方向転換によるコストの差が最適性の原理を壊してしまうため，違う方向から来た経路は区別して考えなければならない．そのため，それらの経路が陽に区別されるよう頂点を複製すればよい．

正規言語制約最短経路問題では，途中の頂点まではどのような経路を「最適」と見なすかが問題である．まず，途中の頂点までの経路にも同じ正規言語制約を課するのは無意味である．たとえば経由頂点指定最短経路問題であれば，最適経路の部分経路がすべて指定された頂点を経由するわけはない．一方，各頂点までの最短の経路を考えるだけでは，正規言語制約を満たさない経路しか得られないかもしれない．これを避けるためには，たとえば経由頂点指定最短経路問題であれば，指定された頂点を経由した経路とそうでないものは区別して考える必要がある．この観察を一般化するために，正規言語がオートマトンの遷移で特徴づけられることに着目する．このオートマトンの状態は最適経路の弁別に十分な情報を与える．ある頂点への経路 x と y をオートマトンが走査すると同じ状態に至るとする．このとき，任意の経路 z について，x を z で延長した経路が受理できるとき，またそのときに限り，y を z で延長した経路も受理できる．よって最適経路の発見には x と y の短い方のみを考えれば十分である．以上から，グラフとオートマトンの積構成を行い経路が状態によって区別されるグラフを構築すればよい，という観察が得られる．

同様のアプローチは Joksch の制約最短経路問題にも適用できる．この問題では，各頂点までの最適経路として重み制約を満たす最短の経路を採用するのが自然に思える．しかしこれは不適切である．多少遠回りだが重みが少ない経路は，短いが重みの大きい辺を利用できる可能性があるため，最終的にはより良い経路を生むかもしれない．このような事態を避けるためには，重みの異なる経路は区別して考える必要がある．実際，経路の重みは最適性の弁別に十分な情報である．よって，各頂点までの経路を重みで区別するよう拡大したグラフ上で最短経路を求めればよい．実際，この帰着は Joksch の動的計画法アルゴリズムと抽象的には同じ処理を行う．

以上見てきたように，最適経路の発見のためには，最適経路を弁別するのに十分な情報を用いて経路を区別すればよい．我々の言語では経路上の再帰関数によって最適経路を定義する．よって，再帰関数の値で経路を区別すれば十分である．より形式的には，再帰関数

$f_1, f_2, \ldots, f_k$ からなる最適経路問合せ仕様に対し，次の性質が成り立つ．ある頂点への経路 x と y が任意の $1 \leq i \leq k$ について $f_i(x) = f_i(y)$ を満たし，かつ x の目的関数値が y のもの以下であるとする．このとき，任意の辺 e および経路 z について，$x \overset{e}{\rightsquigarrow} z$ の目的関数値は $y \overset{e}{\rightsquigarrow} z$ のもの以下であり，また $x \overset{e}{\rightsquigarrow} z$ が制約条件を満たすとき，またそのときに限り，$y \overset{e}{\rightsquigarrow} z$ が制約条件を満たす．この性質は帰納法によりすぐに証明できる．

しかし，上の方法は実用的ではない．まず，我々の言語では目的関数も仕様で定義される 1 関数にすぎず，たとえば目的関数値を制約条件でも用いることができる．そのため，目的関数も最適経路の弁別に必要である．しかし，目的関数値が異なる経路をすべて区別してしまうのでは「目的関数値が比較的小さいものを延長して最適解を得る」ということができず，きわめて効率が悪いアルゴリズムとなってしまう．またアルゴリズムの停止性の問題もある．我々のアプローチは経路のクラス分けに従ってグラフを拡大する．そのため，経路のクラスが無限種類ある場合にはサイズが無限のグラフに対する問題に帰着することになり，アルゴリズムが停止しない可能性がある．

以上の観察をふまえ，有限個のクラスに経路を弁別する．

定義 2　提案言語で記述したプログラム P 中で定義されている再帰関数 f に対し，関数 $\tilde{f}$ を次のとおり定義する．f が真偽値関数ならば $\tilde{f}(x) = f(x)$ である．f が非負整数値関数ならば $\tilde{f}(x) = \min(f(x), u_f)$ である．ここで u_f は以下の 2 条件を満たす最小の非負整数である．

- f の値を直接含む式 exp からなる P 中の不等式 $exp \preceq n$ について，$\preceq \in \{<, \geq\}$ ならば $u_f \geq n$，$\preceq \in \{\leq, \neq, >, =\}$ ならば $u_f > n$ が成り立つ．
- 関数 f_i の定義右辺式が関数 f_j の値を直接含むならば，$u_{f_i} \leq u_{f_j}$ が成り立つ．

関数 $\tilde{f}$ は u_f 以上の値を区別しない点のみが f と異なる．$\tilde{f}$，u_f，f の関係を**図 3** に示す．

記法上の簡便のため関数 $\tilde{\boldsymbol{f}}_P$ を定義する．$\tilde{\boldsymbol{f}}_P$ はプログラム P 中の各 $\tilde{f}_i$ の結果をすべて集めたものである．

$$\tilde{\boldsymbol{f}}_P(x) = (\tilde{f}_1(x), \ldots, \tilde{f}_k(x)) \qquad \text{ただし } f_1(x), \ldots, f_k(x) \text{ は } P \text{ 中のすべての再帰関数}$$

提案アルゴリズムは $\tilde{\boldsymbol{f}}_P$ を用いて経路をクラス分けする．$\tilde{\boldsymbol{f}}_P$ の値域は有限である．そのうえ，以下の補題に示すとおり，$\tilde{\boldsymbol{f}}_P$ は確かに最適経路の弁別に十分な情報を与える．

補題 3　提案言語で記述したプログラム P について，$\tilde{\boldsymbol{f}}_P(x) = \tilde{\boldsymbol{f}}_P(y)$ なる経路 x および y を考える．このとき，以下の 3 つが成り立つ．

(1)　P 中の任意の真偽値式 exp_{bool} について，$exp_{bool}(x) = exp_{bool}(y)$ である．

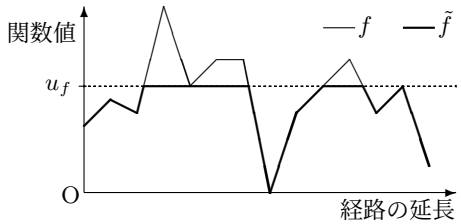

図 3　経路の延長に際しての f と $\tilde{f}$ の変動の様子
Fig. 3　Comparison of values of f and $\tilde{f}$.

(2)　任意の辺 e およびその終点 v について，$\tilde{\boldsymbol{f}}_P(x \overset{e}{\rightsquigarrow} v) = \tilde{\boldsymbol{f}}_P(y \overset{e}{\rightsquigarrow} v)$ である．

(3)　目的関数 f について $f(x) \leq f(y)$ であるとする．このとき，任意の辺 e およびその終点 v について，$f(x \overset{e}{\rightsquigarrow} v) \leq f(y \overset{e}{\rightsquigarrow} v)$ である．

証明　(1)　式の構造に関する帰納法により直截である．

(2)　各再帰関数 f_i を考える．f_i が真偽値関数ならば (1) より明らかである．f_i が非負整数値関数の場合は f_i の定義の右辺式の構造に関する帰納法による．(1) より if 式では同じ分岐が選ばれるため，以下 if 式がないものとする．非自明なのは関数 f_j の呼び出しの場合のみである．$f_j(x) = f_j(y)$ ならば問題ない．$f_j(x) \neq f_j(y)$ なら $\tilde{\boldsymbol{f}}_P(x) = \tilde{\boldsymbol{f}}_P(y)$ より明らかに $\tilde{f}_j(x) = \tilde{f}_j(y) = u_{f_j}$ である．このとき $\tilde{f}_i(x \overset{e}{\rightsquigarrow} v) = \tilde{f}_i(y \overset{e}{\rightsquigarrow} v) = u_{f_i}$ である．なぜなら，整数値式の値は式の構成要素に対して（if 式を除いて）単調増加であり，かつ定義より $u_{f_i} \leq u_{f_j}$ であるからである．

(3)　(1)，(2)，および目的関数に対する制約より，f の定義の右辺式の構造に関する帰納法により証明は直截である．　□

4.2　最適経路問合せアルゴリズム

補題 3 より，各経路はそれらの $\tilde{\boldsymbol{f}}_P$ の値が異なる場合のみ区別すれば十分であることが分かる．これにより，グラフ G 上の最適経路問合せ P は，以下で定義されるグラフ G_P 上の最短経路問合せに帰着できる．

- G_P の頂点は G の頂点 v と $\tilde{\boldsymbol{f}}_P$ の値域上の値 s の組 (v, s) および特殊な頂点 v^+ である．
- G 中の v_1 から v_2 への辺 e および G 中の v_1 への経路 x のそれぞれについて，G_P の頂点 $(v_1, \tilde{\boldsymbol{f}}_P(x))$ から頂点 $(v_2, \tilde{\boldsymbol{f}}_P(x \overset{e}{\rightsquigarrow} v_2))$ への辺 $e_{\tilde{\boldsymbol{f}}_P(x)}$ が存在する．補題 3 の (2) より，この命名規則の下である名前を持つ辺はただ 1 つであることに注意せよ．また，v^+ は

他のすべての頂点への辺を持つ．

- G_P の頂点 (v, s) および辺 e_s に対する基本関数は G での v および e に対するものと同じ値を持つよう定義する．

定理 4　グラフ G に対する図 2 の言語で記述される最適経路問合せ P を考える．P の目的関数を f として，G_P 上の v^+ から始まる経路 $v^+ \overset{e}{\rightsquigarrow} (v_1, s_1) \overset{e_{1,s_1}}{\rightsquigarrow} \cdots \overset{e_{k,s_k}}{\rightsquigarrow} (v_{k+1}, s_{k+1})$ の重みを $f(v_1 \overset{e_1}{\rightsquigarrow} \cdots \overset{e_k}{\rightsquigarrow} v_{k+1})$ と定める．また，G_P の頂点の部分集合 F を $(v, s) \in F \Leftrightarrow (s = \tilde{\boldsymbol{f}}_P(y) \Leftrightarrow y$ は P の制約条件を満たす）と定義する．このとき，G_P 上での v^+ から F のいずれかの頂点への最小重み経路を $x_{G_P} = v^+ \overset{e}{\rightsquigarrow} (v_1, s_1) \overset{e_{1,s_1}}{\rightsquigarrow} \cdots \overset{e_{k,s_k}}{\rightsquigarrow} (v_{k+1}, s_{k+1})$ とすると，$x = (v_1 \overset{e_1}{\rightsquigarrow} \cdots \overset{e_k}{\rightsquigarrow} v_{k+1}) \in [\![P]\!]_G$ が成り立つ．

証明　補題 3 の (1) より，x は P の制約条件を満たす．よって，x が $x^* \in [\![P]\!]_G$ 以下の重みを持つことを示せば十分である，まず，G_P の定義より x^* に対応する G_P 上の経路 $x^*_{G_P}$ が存在する．x^* は制約条件を満たすため，$x^*_{G_P}$ の終点は今考えている G_P 上の最小重み経路問題の終点の 1 つである．よって，x_{G_P} が最小重み経路であること，および G_P 上での経路の重みが G 上での対応する経路の重みと等しいことより，x の重みは x^* 以下である．　□

補題 3 の (3) より，定理 4 で導入した最小重み経路問題は FIFO 条件を満たす．そのため，標準的な最短経路アルゴリズムによってこの問題を解くことができる．以下 $\tilde{\boldsymbol{f}}_P$ の値域の大きさを k とする．目的関数が単調増加，すなわち $f(x) \leq f(x \overset{e}{\rightsquigarrow} v)$ を満たす場合は，ダイクストラ法により $\mathrm{O}(kV \log(kV) + kE)$ 時間で最適経路問合せを達成できる．また，G_P が DAG（Directed Acyclic Graph）をなすときは，トポロジカルソートを用いる標準的なアルゴリズムにより $\mathrm{O}(kE)$ 時間で最適経路問合せを達成できる．k は仕様記述にのみ依存すること，また k は仕様記述の大きさに対しては指数的に大きくなりうることに注意せよ．

効　率　化

計算量は $\tilde{\boldsymbol{f}}_P$ の値域の大きさに依存する．そのためできる限りこれを小さくしたい．G_P の各頂点は $\tilde{\boldsymbol{f}}_P$ の値と G の各頂点からなるため，経路の終点のみから値が定まる関数については $\tilde{\boldsymbol{f}}_P$ の中に含める必要はない．また，その延長が絶対に制約条件を満たさない経路，たとえばある頂点からの経路を求める場合のその頂点以外から始まる経路，に対する計算は無駄である．$\tilde{\boldsymbol{f}}_P$ の値域は有限であるため，ある経路 x に対する $\tilde{\boldsymbol{f}}_P(x)$ の値から $\tilde{\boldsymbol{f}}_P(x \overset{e}{\rightsquigarrow} z)$ がとりうる値は簡単に見積もることができる．各基本関数が任意の値をとる場合にとりうる $\tilde{\boldsymbol{f}}_P$ の値を全列挙すればよい．この情報をもとに，制約条件を真にするような $\tilde{\boldsymbol{f}}_P$ の値を今後とりえない経路を取り除くことで，この無駄をなくすことができる．

4.3　記述例に対する $\tilde{\boldsymbol{f}}_P$ と計算量

以下，3.3 節で示した記述例それぞれについて，$\tilde{\boldsymbol{f}}_P$ の定義を示し，定理 4 に基づく最適経路問合せに要する時間を議論する．G_P の大きさが問合せに要する時間の指標となるため，以下ではこれを見積もる．なお簡便のため v^+ を除いた大きさを考える．

4.3.1　2 点間最短経路

定義より $\tilde{\boldsymbol{f}}_P(x) = (\tilde{cost}(x), to(x), from(x))$ となる．このプログラムは不等式を含まないため $\tilde{cost}(x) = 0$ である．効率化により $from$ の値が偽のものは不要であることが分かる．また，to の値は G の頂点によって一意に定まるためこれを取り除く．以上をふまえると G_P は v^+ を除いて G と同型となる．

4.3.2　3 点間最短経路

$\tilde{\boldsymbol{f}}_P(x) = (\tilde{cost}(x), to(x), from(x), via(x))$ である．効率化をふまえると $\tilde{\boldsymbol{f}}_P$ 中本質的に区別が必要なのは via の値のみであり，G_P は G のたかだか倍の大きさとなる．

4.3.3　乗換コスト付き最短経路

$\tilde{cost}'(x) = 0$ として $\tilde{\boldsymbol{f}}_P(x) = (\tilde{cost}'(x), to(x), from(x), walk(x))$ である．$walk$ のみがグラフを大きくする原因であり，G_P は G のたかだか倍の大きさとなる．

4.3.4　乗換回数制約最短経路

$\tilde{\boldsymbol{f}}_P(x) = (\tilde{cost}(x), to(x), from(x), walk(x), \tilde{transit}(x))$ である．ただし $\tilde{transit}(x) = \min(transit(x), K)$ である．効率化をふまえると，$\tilde{transit}(x) = K$ となる経路 x は考える必要がない．よって，G_P の大きさは G のたかだか $2K$ 倍である．

4.3.5　部分和問題

$\tilde{count}(x) = 0$，$\tilde{sum}(x) = \min(sum(x), W + 1)$ として $\tilde{\boldsymbol{f}}_P(x) = (\tilde{count}(x), \tilde{sum}(x))$ である．入力となる整数値を n 個とすると，グラフ G の大きさは $\mathrm{O}(n)$ であり，G_P の大きさはこの $W + 1$ 倍である．G_P は DAG なので，問合せは $\mathrm{O}(nW)$ 時間で達成できる．

4.4　既知のアルゴリズムとの関係

提案手法は既知の手法の一般化となっており，その漸近計算量はいくつかの問題について既知のアルゴリズムによるものと一致する．たとえば最短経路問題の場合には，G と G_P は同型であるため差は生じない．時間窓制約最短経路問題では，この言語で記述できる問題については，その計算量は Desrochers ら[17] によるものと一致する．各辺の経過時間が正であることにより G_P が DAG となることに注意せよ．方向依存最短経路問題・経由頂点指定最短経路問題についてはそれぞれ Caldwell[8] および Ibaraki[25] のアルゴリズムと等価であ

る．正規言語制約最短経路問題の場合，制約となる正規言語をとらえるのに決定的有限状態オートマトンを用いる点以外は Romeuf[39] のものと等価である．時間依存最短経路問題については Cooke ら[12] のアルゴリズムと等価である．これらの組合せについても同様の議論ができる．たとえば方向依存・時間依存・正規言語制約最短経路問題の場合，オートマトンが決定的か非決定的かの差および扱うことのできる FIFO 条件の範囲の差以外は，Sherali ら[43] のものと等価である．

定理 4 は必ずしも既知の最速のアルゴリズムを導出できるわけではない．たとえば，経由頂点指定最短経路問題に対して Dreyfus[19] のアルゴリズムを，無向グラフに対する最大容量問題に対して Punnen[37] のアルゴリズムを導出することはできない．

5. 最適経路問合せシステム

5.1 システムの設計方針

前章で示した手法を基に，我々は最適経路問合せシステムを試作した．なお，このシステムは第 1 著者のウェブサイト[⋆1]で公開されている．

このシステムは以下の方針に基づいて作られた．

- システムは定理 4 に基づきダイクストラ法または DAG の最短経路アルゴリズムを用いて最適経路を求める．ダイクストラ法を用いる場合，目的関数 f が単調増加，すなわち $f(x) \leq f(x \overset{e}{\rightsquigarrow} v)$，でなければならない．基本関数値が非負であることをふまえると，現実的な例の多くでは目的関数の単調性は成り立つと予想できる．一方，DAG の最短経路アルゴリズムは目的関数の単調増加性を要求しない．
- 最適経路問合せアルゴリズムは C++で実装する．これには主に 3 つの理由がある．C++は広く使われている言語である．C++は低レベルな記述を駆使した非常に効率の良い実装を与えることができる．そして，C++ではグラフ操作ライブラリの事実上の標準である C++ Boost Graph Library[44]（以下 BGL と略記）が利用できる．
- システムはユーザによる最適経路問題の仕様記述から最適経路問合せを行う C++のコードを生成する．理由は実装が簡単でありかつコード生成時に効率化を行うことができるためである．なおコード生成器は Haskell で実装した．システムの概略を**図 4** に示す．

⋆1 http://www.riec.tohoku.ac.jp/~morihata/OPQ.tar.gz

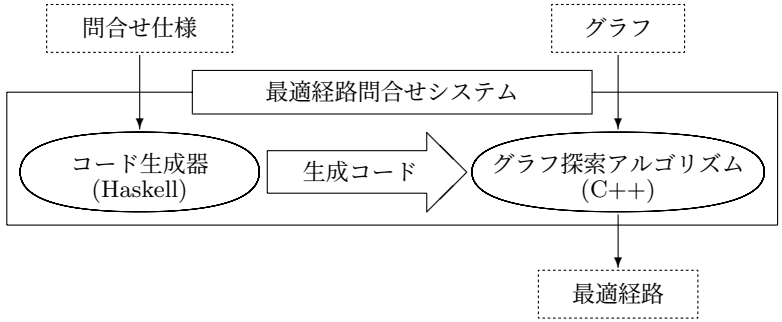

図 4　最適経路問合せシステムの概略
Fig. 4　Overview of the optimal path querying system.

5.2 システムへの入力の設計

システムへの入力は仕様記述とグラフである．グラフは BGL で定義されているグラフ構造をそのまま利用することにした．これにより BGL を用いて記述したプログラムから我々のシステムを簡単に利用できる．

仕様記述のための領域限定言語はほとんど図 2 そのものである．唯一の問題は基本関数の定義方法である．コード生成器は基本的にグラフからは切り離されており，たとえばグラフから辺重みを取り出す方法などを把握できない．そのため，基本関数の定義には`#define` マクロを利用することにした．このマクロは生成されるコード中にそのまま出力される．BGL ではグラフに関する関数はグラフを引数にとるが，マクロの記述を容易にするため，この引数を隠蔽するためのラッパも用意した．マクロの正しさはユーザの責任である．

図 5 に 3.3 節で示した乗換回数制約最短経路問題の記述例を示す．`#define` で始まる行がマクロである．`source`，`target`，`edge_property` は BGL の関数に対するラッパであり，それぞれ辺の始点，終点，そして辺に関連付けられた情報を返す．`v2i` は頂点をその識別番号に割り当てる関数であり，やはり我々のシステムで提供されている．

5.3 効率の良い実装のための工夫

効率の良い最適経路問合せのためにはいくつかの点に留意する必要がある．

5.3.1 拡大グラフ構築の回避

定理 4 では G_P に対し最短経路を計算することを考えているが，G_P は G よりはるかに大きく，メモリに収めるのすら困難である．我々は，各解候補に対し $\tilde{f}_P$ の値を覚えることでこの大きなグラフの構築を避けた．G の同じ頂点への経路であっても $\tilde{f}_P$ の値が違うもの

```
minimize cost(x) s.t. from(x) && to(x) && transit(x) < 20
where

#define START(v) (v==init_vertex)
#define END(v) (v==final_vertex)
#define TRAIN(e) (v2i(source(e))%2 && v2i(target(e))%2)
#define WEIGHT(e) (edge_property(e,&edge_Property::weight))

bool from(v)        = START(v);
     from(x -e-> v) = from(x);
bool to (v)         = END(v);
     to (x -e-> v) = END(v);
int transit()         = 0;
    transit(x -e-> v) = transit(x) + (if walk(x) && TRAIN(e) then 1 else 0);
int cost(v)         = 0;
    cost(x -e-> v) = cost(x) + WEIGHT(e);
bool walk(v)        = true;
     walk(x -e-> v) = !TRAIN(e);
```

図 5　システムへ入力する仕様記述の例（乗換回数制約最短経路問題）
Fig. 5　An example of inputs to the system (the transit-times constrained shortest path problem).

はその値ごとに最短経路を覚えるようダイクストラ法を変更することで，G_P 上での探索を G 上で模倣できる．これは Irnich ら[27] や Sherali ら[43] のアルゴリズムでも用いられている手法である．

5.3.2　プライオリティキューの選択

効率の良い最適経路問合せには効率の良い最短経路アルゴリズムが必要である．効率の良いダイクストラ法の実装に関しては近年 Chen ら[10),11),38)] が詳細に調査し，プライオリティキューとして sequence ヒープ[40] を用いた実現が最速であったと報告している．提案システムもこれをもとに実現されている．なお，sequence ヒープの実装はその発明者である Sanders による非常に効率の良いもの[⋆1]を用いた．

5.3.3　ヒープに挿入する解候補の削減

Sequence ヒープは，キャッシュミスが少ないという長所を持つが，ヒープ中の要素の値の変更は高速に実行できない．そのため，ある頂点に対して既知の経路より短い経路を発見した際には，ヒープ中でのその頂点に対する値を減らすのではなくヒープに新規に要素を挿入することになる．よって通常のダイクストラ法の実装に比べ漸近計算量の観点からはやや悪く，特にメモリを多く消費する．この欠点を補うためにはヒープに挿入する要素数はできる限り減らしたい．

⋆1 http://www.mpi-inf.mpg.de/~sanders/programs/spq/

ダイクストラ法を用いる場合には，制約条件を満たす経路以上の目的関数値を持つ経路は最適解の発見に不要である．そのため，今まで発見した制約条件を満たす経路およびその目的関数値をヒープとは別に覚えておき，この値以上の目的関数値を持つ経路はヒープへ挿入しないことにする．ヒープ中の要素の最小値が制約条件を満たす経路の目的関数値が以上になれば，この経路が最適解である．この効率化を行うと，特に密なグラフでは消費メモリが大いに減少する．

5.3.4　記憶する情報の圧縮

関数 $\tilde{f}_P$ は通常何種類もの値を計算し，これを各解候補に対し記憶するのはかなりのメモリを要する．しかし，$\tilde{f}_P$ の値域はたかだか有限である．しかも，$\tilde{f}_P$ の値域の大きさに計算量が依存するため，$\tilde{f}_P$ の値域がそれほど大きくない場合以外は事実上問合せを処理できない．

この観察をふまえ，我々は，G_P の頂点，すなわち G の頂点と $\tilde{f}_P$ の値の組，をビットフィールドを用いて 1 つの整数値で記憶することにした．すなわち，真偽値関数の結果は 1 ビットで，整数値関数 f_i の結果は $\lceil \log_2(1+u_{f_i}) \rceil$ ビットで管理し，残りのビットで G の頂点を記憶する．なお，G の各頂点を小さなビット数で記憶するため，各頂点に対する 0 から $V-1$ までの識別番号を要求する．

この効率化により，問合せに要するメモリ量は大幅に削減される．さらに次に述べるようにハッシュ値を得るにも好都合である．一方，たとえば整数値が 4 バイトの環境では拡大後のグラフが 2^{32} 以上の頂点を持つ場合は扱えない．しかし，最短経路の計算には頂点数に比例するメモリを要することをふまえると，これほど大きなグラフに対する問合せは現実的ではなく考慮する必要はないと判断した．

5.3.5　到達済み頂点とその距離の記憶

最短経路の計算のためには到達済みの頂点までの最短距離を記憶する必要がある．通常，これは頂点を添字とする配列で管理される．最適経路問合せで同様の実装を行うには G_P の各頂点を添字とした配列が必要だが，我々はそもそも G_P を構成しない．

提案システムでは，配列ではなくハッシュマップを用いて各頂点までの最短距離を管理している．ハッシュマップを効率良く用いるためには適切なハッシュ値が必要であるが，拡大したグラフの頂点は整数値 1 つで管理されているため，これにマスクをかけることで容易にハッシュ値を計算できる．

実装されているハッシュマップは以下のとおり動作する．まず，$\tilde{f}_P$ の値域の大きさを見積もり，適当な非負整数 k を選ぶ．ハッシュマップの初期サイズは入力グラフの頂点数の 2^k 倍とし，ハッシュ値としては頂点の識別番号および $\tilde{f}_P$ 値の適当な k ビットを用いる．ハッ

シュマップの要素数が増加し再ハッシュが起こるたびに，サイズは倍増させ，ハッシュ値は $\tilde{f}_P$ の値をさらに 1 ビット用いる．このとき，ビット数あたりのとりうる値の種類が多い変数から順にハッシュ値として利用する．これにより，軽量かつ実用上比較的衝突が少ないハッシュを実現できる．

5.3.6　最適経路の記憶

最適経路問合せでは，最適な目的関数値ではなく最適経路を求めるが，解候補となる経路をすべて覚えておくのは非効率である．これを避ける標準的な手法は，各頂点への最適経路についてその直前の頂点のみを覚え，最適解発見後にこの情報を用いて最適経路を再構成するものである．提案システムの実装でもこの手法を採用している．

なお，最短経路の場合と違い，最短経路問合せでは 2 頂点間に並行して存在する複数の辺や自己閉路となる辺にも本質的な意義がある．そのため，提案システムでは，並行する辺のうちどれを通ったかも求めなおしている．この計算は，直前の頂点ではなく直前の辺を覚えることで回避できる．しかし，G_P の辺の記憶には G の辺と $\tilde{f}_P$ の値の両方を覚える必要がある．一方，圧縮表現により，G_P の頂点の記憶は $\tilde{f}_P$ の値の記憶と同等である．どの辺を通ったかの確認は通常軽量であり，G の辺を余計に記憶するのに見合う速度向上を生まないと判断し，このような実現とした．

5.4　最適経路問合せ手続きの実現

以上をふまえた最適経路問合せ手続きを**図 6** に示す．これはダイクストラ法を用いるものであり，DAG の最短経路アルゴリズムを用いる場合も同様に実現できる．この手続きの時間計算量は，$\tilde{f}_P$ の値域の大きさを k として，$\mathrm{O}(kE\log(kE))$ である．

また，**図 7** に図 5 のプログラムから生成されるコードを示す．`state` は G_P の頂点を表す共用体である，`init` および `next` は経路の G_P 上での終点を計算する．`HT` はハッシュマップの型である．`iw` および `nw` は経路の重みを計算する．`accept` は経路が制約条件を満たすか否かを判定し，`useful` はその経路の延長が制約条件を満たす経路を生みうるかを調べる．

6. 評 価 実 験

提案システムの効率を確認するため，3.3 節に示した問題について既存のライブラリまたは人手によるプログラムとの比較を行った．なお，コードの生成はいずれの例についても一瞬で完了している．そのため，以下では最適経路問合せの部分についてのみ議論する．実験環境は，CPU が dual quad-core Xeon X5550 2.66 GHz，メモリが 12 GB，コンパイラが GCC 4.5.0，OS が Linux 2.6.32（Ubuntu 10.04）である．なお，この計算機はマルチコア・マルチ CPU であるが，実験に用いたプログラムはいずれも並列計算をまったく行わない．計算時間には入出力に要した時間は含まず，また使用メモリ量にはグラフの記憶に要したメモリ量を含む．

入力グラフを G，最適経路の仕様記述を P，P の目的関数を f とする．以下，(v^*, s^*) は G_P の頂点であり最適経路候補を表現し，w^* はその目的関数値である．W は G_P の頂点までの最短経路の重みを返すハッシュマップである．S は G_P の頂点までの最短経路における直前の頂点を返すハッシュマップである．Q は sequence ヒープであり，目的関数値をキーとし，G_P の頂点を返す．

optimal_path_querying(プログラム P，グラフ G) {
(1)　$w^* := \infty$，$W(\_) = \infty$，$Q = \emptyset$ と初期化する．
(2)　G の各頂点 v について insert(v, v^+) を行う．
(3)　Q が空ならば $[\![P]\!]_G = \emptyset$ である．アルゴリズムを終了する．
(4)　Q 中の最小の重み w を持つ要素 $(w, (v, s)) \in Q$ を取り出す．
(5)　$W((v, s)) \leq w$ ならば，ステップ (3) に戻る．
(6)　$w \geq w^*$ ならば，$x := v^*$ および $(v, s) = (v^*, s^*)$ としステップ (9) に進む．
(7)　終点 v，$\tilde{f}_P(y) = s$，$f(y) = w$ を満たす適当な経路 y を仮想的に考え，v を始点とする各辺 e について，e の終点を v_e として insert$(y \overset{e}{\rightsquigarrow} v_e, (v, s))$ を行う．
(8)　ステップ (3) に戻る．
(9)　$S((v, s)) = v^+$ ならば x が最適経路である．アルゴリズムを終了する．
(10)　$(v', s') = S((v, s))$ とする．終点 v'，$\tilde{f}_P(y) = s'$，$f(y) = W((v', s'))$ を満たす適当な経路 y を擬似的に考え，$\tilde{f}_P(y \overset{e}{\rightsquigarrow} v) = s$ かつ $f(y \overset{e}{\rightsquigarrow} v) = W((v, s))$ 満たす辺 e を発見する．
(11)　$x := v' \overset{e}{\rightsquigarrow} x$ および $(s, v) := (s', v')$ とし，ステップ (9) に戻る．
}

insert(経路 y，G_P の頂点 vs) {
(1)　y の終点を v，$w = f(y)$，$s = \tilde{f}_P(y)$ とする．
(2)　$w^* \leq w$，$W((v, s)) \leq w$，または y の延長が制約条件を満たし得ないならばこの関数から抜ける．
(3)　$W((v, s)) := w$，$S((v, s)) := vs$ とする．
(4)　y が P の制約条件を満たすなら，$w^* := w$，$(v^*, s^*) := (v, s)$ とする．さもなければ Q に $(w, (v, s))$ を挿入する．
}

図 6　最適経路問合せ手続き
Fig. 6　A procedure of optimal path querying.

6.1　最短経路問題の変種に関する実験

最短経路問題とその変種については BGL との比較を行った．BGL には，ダイクストラ法だけでなく，資源制約最短経路問合せ[27] のための関数も実装されている．そのため比較対象として適切であろうと判断した．BGL には仕様から最適経路問合せのためのコードを得る機能はなく，提案システムが生成するコードに対応する部分も人手で記述する必要があ

```
#include <algorithm>

#define START(v) (v==init_vertex)
#define END(v) (v==final_vertex)
#define TRAIN(e) (v2i(source(e))%2 && v2i(target(e))%2)
#define WEIGHT(e) (edge_property(e,&edge_Property::weight))

union state {
    unsigned int hash;
    struct {
        unsigned walk : 1;
        unsigned transit : 5;
        unsigned from : 1;
        unsigned __vertex : (sizeof(int)*8-7);
    } s;
};

inline state init(const vertex &v) const {
    state s;
    s.s.walk = true;
    s.s.transit = std::min(0,20);
    s.s.from = START(v);
    s.s.__vertex=v2i(v);
    return s;
}

inline state next(const state n, const edge &e, const vertex &v) const {
    state s;
    s.s.walk = !TRAIN(e);
    s.s.transit = std::min((n.s.transit+((n.s.walk && TRAIN(e)) ? 1 : 0)),20);
    s.s.from = n.s.from;
    s.s.__vertex=v2i(v);
    return s;
}

typedef hashtable<state,state,7> HT;

inline int iw(const vertex &v) const { return 0; }

inline int nw(const state n, const edge &e, const vertex &v, const int cost) const {
    return (cost+WEIGHT(e));
}

inline bool accept(const state n, const vertex &v) const {
    return ((n.s.from && END(v)) && n.s.transit<20);
}

inline bool useful(const state n) const {
     return (n.s.from && n.s.transit!=20);
}

#undef START
#undef END
#undef TRAIN
#undef WEIGHT
```

図 7　生成されるコードの例（図 5 の入力に対応）
Fig. 7　An example of generated codes, which is obtained from the input in Fig. 5.

表 1　アメリカ道路網グラフの大きさ
Table 1　Sizes of the American road-network data.

	NY	FLA	CAL	LKS	USA-E	USA-W
頂点数	264,346	1,070,376	1,890,815	2,758,119	3,598,623	6,262,104
辺数	733,846	2,712,798	4,657,742	6,885,658	8,778,114	15,248,146

る．そのため，提案システムに比べ，ユーザが最適経路問合せを利用するために求められる知識および労力ははるかに大きい．

ちなみに，BGL のダイクストラ法は relaxed ヒープ[20] を用いて実現してあり，その時間計算量は $O(V \log V + E)$ である．また，資源制約最短経路アルゴリズムはダイクストラ法に似た処理を行い，プライオリティキューとしては二分ヒープを用いている．

実験を行ったのは，2 点間最短経路（以下 SP），3 点間最短経路（3SP），乗換コスト付き最短経路（以下 TRC），乗換回数制約最短経路（以下 TRL）である．TRL では乗換回数は 20 回未満でなければならないとした．比較には，BGL のダイクストラ法を用いた実装（BD-SP）および資源制約最短経路アルゴリズムを用いた実装（問題名を P として BR-P）を用意した．なお，「電車に乗っている」に対応する情報はないため，適当な述語によってこれを代用した．用意したそれぞれの実装について，100 種類の始点および終点（3SP については経由点も）をランダムに生成し，問合せに要した時間の平均値と使用メモリ量の最大値を計測した．

6.1.1　道路網ネットワークに対する最適経路問合せ

まず，9th DIMACS implementation challenge: shortest paths [15),⋆1] のベンチマーク用データであるアメリカ道路網の時間距離のグラフを入力として実験を行った．それぞれのグラフの大きさを**表 1** に示す．

実験の結果を**表 2** に示す．すべての例について，提案システムは BGL での実行速度を大きく上回った．

まず，2 点間最短経路について見る．提案ライブラリは，BGL でのダイクストラ法の実装（BD-SP）に比べ約 2 倍程度早い．これは主に sequence ヒープを用いたことによる改善である．なお，BGL の資源制約最短経路アルゴリズムによる 2 点間最短経路問合せの実装（BR-SP）は，他の 2 つに比べてかなり遅く，また消費メモリ量も多い．

最短経路問題の変種については，提案システムは BGL の資源制約最短経路アルゴリズム

⋆1 http://www.dis.uniroma1.it/~challenge9/

表 2　アメリカ道路網に対する実験結果：平均実行時間（秒）/最悪使用メモリ量（MB）
Table 2　Results for American road networks: average computational time (second)/largest memory use (MB).

	NY	FLA	CAL	LKS	USA-E	USA-W
SP	0.06/77	0.27/317	0.41/478	0.62/699	0.81/900	1.23/1,554
BD-SP	0.12/78	0.41/317	0.85/486	1.15/698	1.63/898	3.34/1,553
BR-SP	0.38/102	1.49/433	3.58/653	4.15/959	5.76/1,235	10.78/2,145
3SP	0.15/87	0.55/366	0.96/539	1.34/787	1.87/1,011	3.69/1,751
BR-3SP	0.88/130	3.60/548	6.42/855	9.73/1,254	13.51/1,619	24.67/2,815
TRC	0.10/83	0.36/350	0.54/509	0.82/744	1.09/955	1.68/1,649
BR-TRC	0.66/111	2.56/466	3.73/712	5.62/1,043	7.73/1,342	14.49/2,333
TRL	1.92/342	2.37/545	3.05/950	7.72/1,831	3.26/1,358	2.22/1,849
BR-TRL	37.99/774	20.15/1,171	24.20/2,163	110.22/5,008	31.12/3,190	17.72/3,396

表 3　ランダム生成グラフの大きさ
Table 3　Sizes of the randomly-generated graphs.

	SMALL	MIDDLE	SPARSE	DENSE	LARGE
頂点数	524,288	1,048,576	2,097,152	524,288	2,097,152
辺数	3,145,728	6,291,456	4,194,304	12,582,912	12,582,912

によるものに比べて一貫してかなり高速であった．特筆すべきは，特に TRL において消費メモリ量に大きな差がある点である．提案システムはビットフィールドによる圧縮表現を用いるため仕様が複雑になっても消費メモリ量はそれほど大きくはならない．一方，BGL のものでは，ユーザがそのようなコードを記述しないかぎり，計算過程で用いる変数が増加すると消費メモリ量が劇的に増加する．

6.1.2　ランダム生成グラフに対する最適経路問合せ

次に，各辺の両端点およびその重みをランダムに生成して得たいくつかのグラフについても同様の実験を行った．なお，辺重みは 1 から 8,192 までの値から一様に選ばれている．ランダム生成したグラフは道路網のグラフに比べ各頂点を少ない辺数で行き来できる．また，今回生成したグラフは辺重みの分散が大きい．以上の要因から，かなり多くの頂点への経路を解候補として検討する必要がある．

実験に用いたグラフの大きさを**表 3** に示す．比較的小さなもの（SMALL），中程度の規模のもの（MIDDLE），辺数の少ないもの（SPARSE），辺数の多いもの（DENSE），そして大きなもの（LARGE）を用意した．

実験結果を**表 4** に示す．なお，LARGE に対する BR-TRL は消費メモリが多すぎて実行

表 4　ランダム生成グラフに対する実験結果：平均実行時間（秒）/最悪使用メモリ量（MB）
Table 4　Results for rondomly-generated graphs: average computational time (second)/largest memory use (MB).

	SMALL	MIDDLE	SPARSE	DENSE	LARGE
SP	0.40/557	0.84/1,105	0.52/577	0.98/1,549	1.26/2,050
BD-SP	0.66/274	1.55/546	1.55/451	1.24/994	2.48/1,090
BR-SP	6.46/486	14.39/971	5.34/633	33.32/2,052	32.84/1,941
3SP	0.91/565	1.86/1,129	1.23/588	2.96/1,985	4.17/2,253
BR-3SP	10.10/544	22.26/1,084	7.70/812	55.24/2,758	51.07/2,165
TRC	0.38/614	1.05/1,202	0.78/624	1.19/1,662	1.57/2,201
BR-TRC	7.76/565	17.06/1,098	21.01/1,266	41.83/2,463	39.64/2,194
TRL	0.99/913	2.72/1,786	6.36/2,328	2.41/2,113	3.28/3,549
BR-TRL	20.78/947	49.03/2,014	13.43/1,552	132.43/4,857	N/A

できなかった．

やはり，提案システムは一貫して BGL を用いた実現より高速であった．ただし，時間計算量から予想されるとおり，頂点数に比べ辺数の多いグラフでは提案システムと BGL のダイクストラ法の速度差は小さい．

今回の例では，道路網の場合とは異なり，提案システムによる最短経路問合せは比較的多くのメモリを消費している．これは，ランダム生成したグラフの頂点間距離の近さから，同一頂点までの経路が多数ヒープ中に存在しているためである．しかし，圧縮表現の利用やヒープに挿入する要素数の削減などの効率化の結果，密なグラフや大きなグラフに対する問合せであっても比較的現実的なメモリ消費量に抑えることに成功している．

6.1.3　考　　察

以上の実験結果において特に注目すべきは以下の 2 点である．

まず，BGL の最短経路問合せ関数（BD-SP）よりも本システムによる問合せが高速であった．漸近計算量は BGL のものの方が良いのだが，sequence ヒープは relaxed ヒープに比べキャッシュミスが少ないため高速な処理が達成されている．これは，一般的な手法を用いるオーバヘッドよりもむしろ最短経路を発見するアルゴリズムの速度の方が重要であることを示唆している．

また，BGL での資源制約最短経路の計算（BR-SP）が最短経路のもの（BD-SP）に比べかなり遅かったのに対し，本システムではやや複雑な最適経路問合せも比較的小さなオーバヘッドで実現できている．これは，最適経路問合せに特化したいくつもの効率化を行った効果である．特に，複雑な仕様に対してはメモリの消費量などに注意が必要であるが，人手で

表 5　部分和問題に関する実験の結果：平均実行時間（秒）/最悪使用メモリ量（MB）．目標値は 8,192 に固定
Table 5　Results for the subset sum problems: average computational time (second)/largest memory use (MB). The objective value is 8,192.

整数値の数	1,024	2,048	4,096	8,192	16,384
提案システム	0.39/251	0.78/506	1.56/1,020	3.13/2,044	6.27/4,095
手実装	0.04/33	0.08/65	0.15/129	0.28/257	0.56/514

表 6　部分和問題に関する実験の結果：平均実行時間（秒）/最悪使用メモリ量（MB）．整数値は 1,024 個に固定
Table 6　Results for the subset sum problems: average computational time (second)/largest memory use (MB). The number of input integers is 1,024.

目標値	8,192	16,384	32,768	65,536	131,072
提案システム	0.39/251	0.74/484	1.36/907	2.33/1,572	3.14/2,093
手実装	0.04/33	0.08/65	0.16/133	0.29/261	0.56/517

の効率化は煩雑でありバグを埋め込みやすい．提案システムでは効率化されたコードを自動的に得ることができる．

6.2　部分和問題への適用

最後に，提案システムを部分和問題に適用した結果について報告する．この問題は DAG の最短経路アルゴリズムに基づく実装で実行した．入力となる整数値は 1 から 255 の範囲でランダムに選んだ．比較対象としては標準的な動的計画法アルゴリズムを人手で実装したものを用意した．

我々が行った 2 つの実験の結果を**表 5** と**表 6** に示す．前者は目標とする値 W を 8,192 に固定し入力とする整数の個数を変動させたもの，後者は入力とする整数の個数を 1,024 に固定し目標値 W を変動させたものである．いずれも 10 種類の入力を生成し，計算に要した時間の平均値と使用メモリ量の最大値を計測した．

パラメータによって差があるものの，提案システムは人手による実装に比べ時間・メモリともに数倍から十数倍悪かった．部分和問題は本来かなり単純な問題であり，グラフのような複雑な構造を導入する必要はない．そのため最適経路問題への帰着はかなりのオーバヘッドを生んでいる．しかし，提案システムは整数値の数および目標値に対しておおよそ線形の実行時間で解を求めている．ユーザがアルゴリズムや効率をまったく気にすることなく問合せを行えることを考えれば，定数倍程度の速度差は気にならない状況では提案システムの利用は有意義であると考えられる．また，プロトタイピングの段階では提案システムを利用し，大規模な入力を扱う段階に達したとき，改めてボトルネックのみを人手によって書き直す，というような利用法も考えられる．

7. 関 連 研 究

我々はグラフを拡大することで最適経路問合せを最短経路問題に帰着する手法を示した．この手法は有限状態オートマトンの積構成に対応する．積構成の利用は特に正規表現経路問合せ[14),22),32),33),39)] の文脈ではよく知られた手法である．正規表現は表現力が高く，またユーザにとっても直感的で扱いやすい．しかし，正規表現で数値を扱うのは困難である．本研究では，数値を比較的自然に扱うことができ，かつ再帰関数が有限状態オートマトンに対応するように領域限定言語を設計した．この点が既存の研究との主な差違である．

本研究の背景にあるのは，動的計画法，最短経路問題，そして有限状態オートマトンの関係である．Bellman[4)] は動的計画法を定式化し，動的計画法と最短経路問題との関連について述べた．Karp ら[29)] は有限状態オートマトンを用いた動的計画法の定式化を提案し，問題の仕様からある条件を満たす有限状態オートマトンを構成することによって動的計画法アルゴリズムを導出できることを示した．Ibaraki[26)] は Karp ら手法に基づき導出された動的計画法の解は最短経路アルゴリズムによって効率良く計算できることを示した．以上をふまえ，我々は，Karp らの議論した有限状態オートマトンを自動的に構成し，Ibaraki の結果に従って最短経路問題へと帰着することを意図した．積構成は有限状態オートマトンの構成にあたって用いた手法の 1 つにあたる．

以上述べたとおり，本研究の要点は領域限定言語の設計にある．これには，最大マークづけ問題の解法に関する研究[1),5),6),41),42)] を参考にした．Sasano ら[41),42)] は，問題の仕様を構造上の原始再帰関数で記述することで，その仕様から最大マークづけ問題の解法が機械的に導出できることを示した．しかし彼らの枠組みは，最大マークづけ問題以外には適用できない，木構造しか扱うことができない，真偽値関数しか仕様記述に用いることができない，などの制約がある．Bird[5)] は Sasano らの成果を抽象化し定式化し直した．Bird の結果は，木構造上の原始再帰関数であれば真偽値関数以外であっても同様の手法が利用できることを示唆している．これらをふまえ，Morihata[34)] は最大マークづけ問題以外にも利用できまた整数値なども利用できるよう一般化を行い，その結果を基に動的計画法アルゴリズムを自動導出するライブラリを実装した．しかし，この手法でも扱えるのは木構造でありグラフ構造は扱うことができない．グラフ構造では，木構造と違い，一般に解の候補が有限に収まらない場合が多く，そのため仕様記述に用いる再帰関数の値域の有限性が重要となる．整数値関数を用いることができ，かつ有限値域の関数で記述された仕様へ機械的に変形できるよう仕様記述言語を設計した点が本研究の技術上の特徴である．

8. 結論と今後の課題

本論文では，領域限定言語に基づいた最適経路問合せアルゴリズムとその効率の良い実装について述べた．提案手法は最適経路問題に関するいくつかの問題クラスおよびその組合せを扱うことでき，計算量の保証されたアルゴリズムを得ることができる．さらに，適切な実装を行うことで，既存のライブラリを上回る速度での最適経路問合せを実現できた．

本研究では扱わなかった最適経路問題の類型はいくつもある．まず，全点間最適経路については理論上は容易に拡張できる．拡大されたグラフ G_P 上で全点間最短経路を解けばよい．同様にして，k 番目最適経路問題も G_P 上の k 番目最短経路問題へと帰着することができる．しかし，これらの効率の良い実装についてはさらなる議論が必要であると思われる．特に G_P を陽に構成することなく問合せを実現するのは簡単ではないと予想している．また，提案手法では時間依存最短経路問題については一部しか扱えない．Orda や Rom の示した FIFO 条件の一般化をうまく領域限定言語に取り入れるのは興味深い課題だと考えている．

提案手法はダイクストラ法や DAG の最短経路アルゴリズムを用いて最適経路問合せを行っているが，最短経路は A*探索によりさらに高速に求められることが多い．A*探索のためのヒューリスティック関数を仕様記述から構成することができればより高速な最適経路問合せが期待できる．

本研究では文脈自由言語制約最短経路問題は扱わなかったが，これも理論的には提案手法と似た手法で扱うことができる．提案手法ではグラフと仕様記述を有限状態オートマトンと見なし積構成を行った．文脈自由言語制約最短経路問題では，仕様記述を文脈自由文法の形で記述し，有限状態オートマトンと文脈自由文法の積構成を行えばよい．結果，文脈自由文法の最小重み導出を求める問題に帰着されるが，これは Knuth[30] の一般化ダイクストラ法によって解くことができる．また，同様の手法でプッシュダウンシステムに対する最適実行トレースなども求めることができる．

最短経路問題や最適経路問題は多くの応用を持つ．提案手法によって具体的な応用を記述できるか，また現実的な時間で解くことができるか，などについてはまたまだ調査が必要である．また，本研究で示した領域限定言語がいままで知られていなかった最適経路問題の応用の発見に役立つことも期待している．

謝辞　研究の初期に最適経路問題を考えることを提案してくださった国立情報学研究所の胡振江氏に感謝する．また，研究の発展および論文の改善に有益な助言をしてくださった PPL2008，ICFP2008，および情報処理学会論文誌「プログラミング」の査読者の皆様に感謝する．

参 考 文 献

1) Arnborg, S., Lagergren, J. and Seese, D.: Easy Problems for Tree-Decomposable Graphs, *Journal of Algorithms*, Vol.12, No.2, pp.308–340 (1991).
2) Barrett, C.L., Bisset, K., Jacob, R., Konjevod, G. and Marathe, M.V.: Classical and Contemporary Shortest Path Problems in Road Networks: Implementation and Experimental Analysis of the TRANSIMS Router, *Algorithms — Proc. ESA 2002, 10th Annual European Symposium*, Rome, Italy, September 17-21, 2002, Lecture Notes in Computer Science, Vol.2461, pp.126–138, Springer (2002).
3) Barrett, C.L., Jacob, R. and Marathe, M.V.: Formal-Language-Constrained Path Problems, *SIAM Journal on Computing*, Vol.30, No.3, pp.809–837 (2000).
4) Bellman, R.: *Dynamic Programming*, Princeton University Press (1957).
5) Bird, R.S.: Maximum marking problems, *Journal of Functional Programming*, Vol.11, No.4, pp.411–424 (2001).
6) Borie, R.B., Parker, R.G. and Tovey, C.A.: Automatic Generation of Linear-Time Algorithms from Predicate Calculus Descriptions of Problems on Recursively Constructed Graph Families, *Algorithmica*, Vol.7, No.5&6, pp.555–581 (1992).
7) Brodal, G.S. and Jacob, R.: Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries, *Electronic Notes in Theoretical Computer Science*, Vol.92, pp.3–15 (2004).
8) Caldwell, T.: On finding minimum routes in a network with turn penalties, *Comm. ACM*, Vol.4, No.2, pp.107–108 (1961).
9) Carré, B.E.: An algebra for network routing problems, *Journal of the Institute of Mathematics and Its Applications*, Vol.7, No.3, pp.273–294 (1971).
10) Chen, M., Chowdhury, R.A., Ramachandran, V., Roche, D.L. and Tong, L.: Priority Queues and Dijkstra's Algorithm, Technical Report TR-07-54, Departments of Computer Sciences, the University of Texas at Austin (2007).
11) Chowdhury, R.A.: Cache-efficient Algorithms and Data Structures: Theory and Experimental Evaluation, Ph.D. Thesis, Department of Computer Sciences, Univerity of Austin (2007).
12) Cooke, K.L. and Halsey, E.: The shortest route through a network with time-dependent internodal transit times, *Journal of Mathematical Analysis and Applications*, Vol.14, No.3, pp.493–498 (1966).
13) Cormen, T.H., Stein, C., Rivest, R.L. and Leiserson, C.E.: *Introduction to algorithms, 2nd edition*, MIT Press, Cambridge, MA, USA (2001).

14) de Moor, O., Lacey, D. and Wyk, E.V.: Universal Regular Path Queries, *Higher-Order and Symbolic Computation*, Vol.16, No.1–2, pp.15–35 (2003).
15) Demetrescu, C., Goldberg, A.V. and Johnson, D.S. (Eds.): *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, American Mathematical Society (2009).
16) Deo, N. and yin Pang, C.: Shortest-path algorithms: Taxonomy and annotation, *Networks*, Vol.14, No.2, pp.275–323 (1984).
17) Desrochers, M. and Soumis, F.: A Generalized permanent labeling algorithm for the shortest path problem with time windows, *INFOR*, Vol.26, pp.191–212 (1988).
18) Dijkstra, E.W.: A Note on Two Problems in Connexion with Graphs, *Numerische Mathematik*, Vol.1, pp.269–271 (1959).
19) Dreyfus, S.E.: An appraisal of some shortest-path algorithms, *Operations Research*, Vol.17, pp.548–566 (1969).
20) Driscoll, J.R., Gabow, H.N., Shrairman, R. and Tarjan, R.E.: Relaxed Heaps: An Alternative to Fibonacci Heaps with Applications to Parallel Computation, *Comm. ACM*, Vol.31, No.11, pp.1343–1354 (1988).
21) Eppstein, D.: Finding the k Shortest Paths, *SIAM Journal on Computing*, Vol.28, No.2, pp.652–673 (1998).
22) Flesca, S., Furfaro, F. and Greco, S.: Weighted path queries on semistructured databases, *Information and Computation*, Vol.204, No.5, pp.679–696 (2006).
23) Fredman, M.L. and Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM*, Vol.34, No.3, pp.596–615 (1987).
24) Hopcroft, J.E., Motwani, R. and Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation, 3rd Edition*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2006).
25) Ibaraki, T.: Algorithms for obtaining shortest paths visiting specified nodes, *SIAM Review*, Vol.15, No.2, pp.309–317 (1973).
26) Ibaraki, T.: Solvable classes of discrete dynamic programming, *Journal of Mathematical Analysis and Applications*, Vol.43, No.3, pp.642–693 (1973).
27) Irnich, S. and Desaulniers, G.: Shortest Path Problems with Resource Constraints, *Column Generation*, Desaulniers, G., Desrosiers, J. and Solomon, M.M. (Eds.), chapter 2, pp.33–65, Springer (2005).
28) Joksch, H.C.: The shortest route problem with constraints, *Journal of Mathematical Analysis and Applications*, Vol.14, No.2, pp.191–197 (1966).
29) Karp, R.M. and Held, M.: Finite-State Processes and Dynamic Programming, *SIAM Journal on Applied Mathematics*, Vol.15, No.3, pp.693–718 (1967).
30) Knuth, D.E.: A Generalization of Dijkstra's Algorithm, *Information Processing Letters*, Vol.6, No.1, pp.1–5 (1977).
31) Korkmaz, T. and Krunz, M.: Multi-Constrained Optimal Path Selection, *Proc. IEEE INFOCOM 2001, The Conference on Computer Communications, 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, 22-26 April 2001, Anchorage, Alaska, USA, pp.834–843, IEEE (2001).
32) Liu, Y.A., Rothamel, T., Yu, F., Stoller, S.D. and Hu, N.: Parametric regular path queries, *Proc. ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation 2004*, Washington, DC, USA, June 9-11, 2004, pp.219–230, ACM (2004).
33) Mendelzon, A.O. and Wood, P.T.: Finding Regular Simple Paths in Graph Databases, *SIAM Jornal on Computing*, Vol.24, No.6, pp.1235–1258 (1995).
34) Morihata, A.: A Short Cut to Optimal Sequences, *New Generation Computing* (2010). To appear.
35) Morihata, A., Matsuzaki, K. and Takeichi, M.: Write it Recursively: A Generic Framework for Optimal Path Queries, *Proc. 2008 ACM SIGPLAN International Conference on Functional Programming, ICFP 2008*, Sep. 22-24, 2008, Victoria, BC, Canada, pp.169–178, ACM (2008).
36) Orda, A. and Rom, R.: Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Length, *J. ACM*, Vol.37, No.3, pp.607–625 (1990).
37) Punnen, A.P.: A linear time algorithm for the maximum capacity path problem, *European Journal of Operational Research*, Vol.53, No.3, pp.402–404 (1991).
38) Ramachandran, V.: Cache-Oblivious Computation: Algorithms and Experimental Evaluation, *2007 International Conference on Computing: Theory and Applications* (*ICCTA 2007*), 5-7 Mar. 2007, Kolkata, India, pp.20–26, IEEE Computer Society (2007).
39) Romeuf, J.-F.: Shortest Path Under Rational Constraint, *Information Processing Letters*, Vol.28, No.5, pp.245–248 (1988).
40) Sanders, P.: Fast Priority Queues for Cached Memory, *ACM Journal of Experimental Algorithmics*, Vol.5, p.7 (2000).
41) Sasano, I., Hu, Z., Takeichi, M. and Ogawa, M.: Make it practical: A generic linear-time algorithm for solving maximum-weightsum problems, *Proc. 5th ACM SIGPLAN International Conference on Functional Programming, ICFP'00*, ACM, pp.137–149 (2000).
42) Sasano, I., Ogawa, M. and Hu, Z.: Maximum Marking Problems with Accumulative Weight Functions, *Theoretical Aspects of Computing — Proc. ICTAC 2005, 2nd International Colloquium*, Hanoi, Vietnam, 17-21 Oct., 2005, pp.562–578, Springer (2005).
43) Sherali, H.D., Jeenanunta, C. and Hobeika, A.G.: The approach-dependent, time-dependent, label-constrained shortest path problem, *Networks*, Vol.48, No.2, pp.57–

67 (2006).
44) Siek, J.G., Lee, L.-Q. and Lumsdaine, A.: *The Boost Graph Library: User Guide and Reference Manual*, Addison-Wesley (2001).
45) Villeneuve, D. and Desaulniers, G.: The shortest path problem with forbidden paths, *European Journal of Operational Research*, Vol.165, No.1, pp.97–107 (2005).

(平成 22 年 9 月 28 日受付)
(平成 22 年 12 月 1 日採録)

森畑　明昌（正会員）

1981 年生．2004 年東京大学工学部計数工学科卒業．2006 年同大学大学院情報理工学系研究科修士課程修了，2009 年同博士課程修了，同年日本学術振興会特別研究員を経て，2010 年東北大学電気通信研究所助教となり現在に至る．博士（情報理工学）．プログラム変換，アルゴリズム導出，並列プログラミング等に興味を持つ．日本ソフトウェア科学会会員．

松崎　公紀（正会員）

1979 年生．2001 年東京大学工学部計数工学科卒業．2003 年同大学大学院情報理工学系研究科修士課程修了．2005 年同研究科博士課程中退．同年より同研究科助手，2007 年より助教，2009 年より高知工科大学情報学群准教授となり現在に至る．博士（情報理工学）．並列プログラミング，アルゴリズム導出等に興味を持つ．日本ソフトウェア科学会会員．

武市　正人（正会員）

1948 年生．1972 年東京大学工学部助手，講師，電気通信大学講師，助教授，東京大学工学部助教授を経て，1993 年東京大学大学院工学系研究科教授（情報処理工学講座），2001 年より同大学院情報理工学系研究科教授，現在に至る．2003 年より日本学術会議会員．工学博士．プログラミング言語，関数プログラミング，構造化文書処理の研究・教育に従事．日本ソフトウェア科学会，日本応用数理学会，ACM 各会員．