日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

効率の良い最適経路問合せシステムの実装

森畑 明昌

グラフ中から最適な経路を発見する「最適経路問合せ」は，経路探索やネットワーク解析など実用上様々な応用がある．そのため，最適経路問合せを効率良く行う手法が求められている．本発表では，広い範囲の最適経路問合せを効率よく達成するシステムの実装について報告する．本システムの特徴は，最適経路の仕様を記述するための領域限定言語をユーザに提供し，ユーザの記述をもとに適切なプログラムを生成する点にある．これにより，ユーザはアルゴリズムに関する知識なしに広い範囲の最適経路問合せを行うことができる．さらに，最短路問題の実装に関する既知の成果を応用し効率的な実装を行うことにより，本システムは既存のライブラリでの実装に比べ効率を高めることに成功した．

1 はじめに

本論文では，グラフ中からある基準の下で最適な経路を求める「最適経路問合せ」を考える．例えば，特定の 2 点間の経路の中で最短のものを求める「2 点間最短路問題」やグラフ中の全点を 1 回ずつ訪問する経路の中で最短のものを求める「巡回セールスマン問題」などが良く知られた例である．他にも，2 点間の経路で重みが一定以下のものの中で最短のものを求める「重み制約付き 2 点間最短路問題」[12]，2 点間の経路でそのラベルが特定の正規言語に属するものの中で最短のものを求める「正規言語制約付き 2 点間最短路問題」[20] など，様々な問題が考えられ，盛んに研究されてきた [2,3,7,10,11,17,18,22,24].

最適経路問合せは多くの実用的な応用を持つため重要である．まず，経路探索システムでの利用 [1] はもっとも直截な応用である．例えば，鉄道での乗換探索であれば,「乗換回数のできる限り少ないもの」や「一定時間以内に到着するものの中でもっとも費用が安いもの」など，ユーザからの様々な要請に応えることが求められる．最適経路問合せでは，これらを統一的に扱うことができる．グラフ構造データベースへの問合せ [9,15] も応用の一つである．グラフ構造データベースに対しては経路をもとに問合せを行うことが一般的であるが，グラフが閉路を持つ場合には問合せにマッチする経路は一般には有限個に収まらない．そのため，問合せにマッチする経路の中から何らかの意味で最適なものを選ぶ必要がある．また，ネットワークの解析にも最適経路問合せは用いられる [13]．ネットワークが所望の性質を満たしているか，例えば最大遅延が一定時間以下かつホップ数が一定以下の経路が存在するか，などの確認において，最適経路問合せは基本的な処理として用いられている．

発表者は以前の研究 [16] で，領域限定言語に基づく最適経路問合せ手法を提案した．この手法は，ユーザにアルゴリズムに関する知識を要求しない，広い範囲の最適経路問合せを扱うことができる，得られるアルゴリズムが既存のいくつかの最適経路問合せアルゴリズムの一般化になっている，などの長所を持つ．しかし，この手法ほど広い範囲の最適経路問合せを効率よく実装する試みは発表者の知る限り今までになかった．

本発表では，[16] に基づいた効率の良い最適経路問合せシステムの実現について報告する．発表者は 2 点

Implementation of Efficient Optimal Path Querying System

Akimasa Morihata, 東北大学電気通信研究所, Research Institute of Electrical Communication, Tohoku University.

間最短路問題の実装に関する最新の成果 [4, 19] を参考にし，さらに最適経路問合せに特徴的ないくつかの点について効率化を加えることにより，既存の実現に比べてはるかに高速な問合せを実現した．

2 領域限定言語に基づく 最適経路問合せシステムの設計

まず，今回提案する最適経路問合せシステムの設計について述べる．なお，システムのソースコードは発表者のウェブサイト†1から入手できる．

提案システムは領域限定言語によって記述された最適経路問合せの仕様記述を元に C++のライブラリを生成するものとなっている．このライブラリは，C++の標準的なグラフアルゴリズムの実装である boost graph library [23] 中で定義されているグラフを入力とし，最適経路を返す関数からなる．この設計には使いやすさと効率の両面から理由がある．

- 最適経路問合せはシステムの一部として用いられることが多い．そのため，単体動作するシステムとして用意するのではなく，他言語からも比較的利用しやすい C++のライブラリとして実現するのは有意義である．
- 一般的なアルゴリズムにはオーバーヘッドが生じがちであるが，プログラム生成時に効率化を行うことで，その多くを取り除くことができる．

3 最適経路問合せのための領域限定言語

3.1 構文文法

提案システムで用いられる最適経路問合せの記述のための領域限定言語の構文文法を図 1 に示す．

プログラムは最小化すべき目的関数と満たさなければならない制約条件からなり，それらは頂点や辺に対する基本関数と経路に対する再帰関数で定義されている．基本関数は C/C++のマクロとして直接定義する．経路は頂点と辺の列であり，$x \overset{e}{\rightsquigarrow} v$ は「経路 x を辺 e を用いて頂点 v まで延長した経路」を意味する．再帰関数の定義ではそれぞれの型に合わせて標準的な演算を用いることができる．減算ができないこと，整数定数として非負整数しか用いることができないこと，そして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注意せよ．

この構文文法に加え，目的関数には付加的な制約がある．目的関数は単調増加，すなわち $f(x \overset{e}{\rightsquigarrow} v) \leq f(x)$ を満たさねばならず，またその定義中では，条件式の条件部分以外に他の関数の呼び出しを含んではならない．これらの条件は，効率の良い最適経路問合せの実現のために要請されている．

3.2 記述例

2 点間最短路

もっとも単純な最適経路問合せは 2 点間最短路である．なお，`-e->`を $\overset{e}{\rightsquigarrow}$ の ASCII 文字による記述として用いている†2．

```
maximize cost(x) s.t. start(x) && end(x)
where
#define START(v) (v==init_vertex)
#define END(v) (v==final_vertex)
#define WEIGHT(e) \
  (edge_property(e,&edge_Property::weight))
bool start(v)        = START(v);
  |  start(x -e-> v) = start(x);
bool end(v)          = END(v);
  |  end(x -e-> v)   = END(v);
int cost(v)          = 0;
 |  cost(x -e-> v)   = cost(x) + WEIGHT(e);
```

2 点間最短路を求めるには，その始点と終点がそれぞれ適切な頂点である経路の中で最短のものを求めれば良い．これを達成するために，3 つの基本関数と 3 つの再帰関数が用意されている．

基本関数 `START` と `END` は，与えられた頂点がそれぞれ始点または終点であるかを確認する．変数 `init_vertex` および `final_vertex` は，この仕様記述から生成されるライブラリを用いる C++プログラムの側から大域変数として提供されることが想定されている．また，基本関数 `WEIGHT` はシステムの提供する補助関数 `edge_property` を用いて，boost graph

†1 `www.riec.tohoku.ac.jp/~morihata/OPQ.tar.gz`

†2 なお，\ を用いた行連結は実際のシステムではサポートされていない．

$$
\begin{array}{rcll}
Prog & ::= & \texttt{minimize}\ f_{int}(x)\ \texttt{s.t.}\ Exp_{bool}\ \texttt{where}\ Decl \cdots Decl & \{\ \text{プログラム}\ \} \\
Decl & ::= & \#\texttt{define}\ \text{<string>} & \{\ \text{基本関数定義}\ \} \\
 & | & \texttt{int}\ f_{int}(v) = Exp_{int}; \quad | \ f_{int}(x \overset{e}{\rightsquigarrow} v) = Exp_{int}; & \{\ \text{整数値関数定義}\ \} \\
 & | & \texttt{bool}\ f_{bool}(v) = Exp_{bool}; \quad | \ f_{bool}(x \overset{e}{\rightsquigarrow} v) = Exp_{bool}; & \{\ \text{真偽値関数定義}\ \} \\
Exp_{int} & ::= & \text{<unsigned\_integer>} \mid m(v,e) \mid Exp_{int} + Exp_{int} \mid Exp_{int} * Exp_{int} & \{\ \text{整数値式}\ \} \\
 & | & \texttt{max}(Exp_{int},Exp_{int}) \mid \texttt{min}(Exp_{int},Exp_{int}) & \\
 & | & f_{int}(x) \mid \texttt{if}\ Exp_{bool}\ \texttt{then}\ Exp_{int}\ \texttt{else}\ Exp_{int} & \\
Exp_{bool} & ::= & \texttt{true} \mid \texttt{false} \mid m(v,e) \mid \texttt{!}Exp_{bool} \mid Exp_{bool}\ \texttt{\&\&}\ Exp_{bool} \mid Exp_{bool}\ \texttt{||}\ Exp_{bool} & \{\ \text{真偽値式}\ \} \\
 & | & f_{bool}(x) \mid Exp_{int} \preceq \text{<unsigned\_integer>} & \\
\preceq & ::= & \texttt{<=} \mid \texttt{>=} \mid \texttt{==} \mid \texttt{!=} \mid \texttt{<} \mid \texttt{>} & \{\ \text{比較演算子}\ \}
\end{array}
$$

図 1 最適経路問合せの記述のための領域限定言語.x はパス変数，v は頂点変数，e は辺変数，f_{int} は整数値関数名，f_{bool} は真偽値関数名である.$m(e,v)$ は基本関数であり，実際にはマクロの呼び出しで表現される.

library で提供されているグラフ構造から，辺 `e` のプロパティ`weight` に関連付けられた値を取り出している．このような，入出力の扱いやグラフ中の値の取り出し方は，最適経路問合せシステムが直接知ることのできない．そのため，C++のプログラムとして直接記述され，また生成されるプログラム中にもそのまま出力される．

これらの基本関数を用いることで，再帰関数は簡単に定義できる．再帰関数 `start` と `end` はそれぞれ始点と終点を確認し，`cost` は辺重みの和を求める．

乗換回数制限最短路

やや複雑な例として，電車に乗車する回数が高々20 回以下と制限された状況で 2 点間の最短路を求める問題を考える．

```
minimize cost(x)
s.t. start(x) && end(x) && n_trans(x) < 20
where
#define START(v) (v==init_vertex)
#define END(v) (v==final_vertex)
#define WEIGHT(e) (...)
#define TRAIN(e) (...)
bool start(x -e-> v) = start(x);
  |  start(v) = START(v);
bool end (x -e-> v) = END(v);
  |  end (v) = END(v);
int n_trans(x -e-> v) = n_trans(x) +
    (if walk(x) && TRAIN(e) then 1 else 0);
  | n_trans(v) = 0;
int cost(x -e-> v) = cost(x) + WEIGHT(e);
  | cost(v) = 0;
bool walk(x -e-> v) = !TRAIN(e);
  |  walk(v) = true;
```

先の例に比べ，今回は電車に乗っているか否かを調べる基本関数 `TRAIN` が導入されている．なお，`TRAIN` の定義は適切に与えられているものとし，ここでは省略する．これを用い，再帰関数 `walk` は直前に電車に乗っていなかったことを確認し，`n_trans` は電車に乗り込んだ回数を数えている．

4 最適経路問合せアルゴリズム

アルゴリズムは先行研究である [16] のものに従い，ここでは簡単な説明のみを行う．

いま，$\{h_1,\ldots,h_n\}$ を仕様記述中に現れる整数値関数の集合，$\{p_1,\ldots,p_m\}$ をプログラム中に現れる真偽値関数の集合とする．また，u_{h_i} を関数 h_i の値が比較に用いられる比較演算子の右オペランドの最大値とする．なお，そのような比較式が存在しない場合には $u_{h_i} = -\infty$ とする．さらに，関数 $\hat{h}_j$ を $\hat{h}_j(x) = \min(h_j(x), u_{h_i} + 1)$ とする．これらを用い，関数 $state$ を以下で定義する．

$$state(x) = (\hat{h}_1(x),\ldots,\hat{h}_n(x),p_1(x),\ldots,p_m(x))$$

関数 $state$ を用いることで，グラフ G 上の最適経

路問合せは，以下の最短経路問合せに帰着される．

- 対象とするグラフの頂点は G の頂点 v と $state$ の値域上の値 s の組 (v, s) である．
- 頂点 (v_1, s_1) から頂点 (v_2, s_2) への辺は，v_1 から v_2 への辺 e が G に存在し，かつ $state(x) = s_1$ なる任意の経路 x について $state(x \overset{e}{\rightsquigarrow} v_2) = s_2$ が成り立つときに存在し，その重みは目的関数を f として $f(x \overset{e}{\rightsquigarrow} v_2) - f(x)$ である．
- 始点は任意の頂点である．
- 終点は，制約条件を満たす経路 x にが存在するような全ての頂点 $(v, state(x))$ である．

この最短経路問合せは 2 点間最短路を求めるアルゴリズムによって容易に実現できる．これより以下を得る．

定理 1 ([16])**.** グラフ G に対する図 1 の言語で記述される最適経路問合せは，G に対する 2 点間最短路問合せの時間を $T(|G|)$，$state$ の値域の大きさを k として，$O(T(k|G|))$ で達成できる． □

例えば，フィボナッチヒープを用いた Dijkstra アルゴリズムを用いれば，頂点数 V，辺数 E のグラフに対し，$state$ の値域の大きさを k として，$O(kV \log(kV) + kE)$ 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．

これに加え，[16] では，不要な解候補を考慮に入れることをできる限り避ける効率化が提案されていが，ここでは詳細は述べない．提案システムでは，明らかに不要な解候補を除去する最適化 ([16] の補題 4.5) を実装している．

5 最適経路問合せシステムの実装

定理 1 より，効率の良い最短経路問合せアルゴリズムを用いれば効率よく最適経路問合せを実現できることが分かる．効率の良い最短経路問合せについては，近年 Ramachandran ら [4,19] が詳細に調査しており，Dijkstra アルゴリズムの decrease-key を用いない変種をプライオリティキューとして Sanders の sequence heap [21] を用いて実現したものが最速であることが報告されている．提案システムもこれを元に実現されている．なお，sequence heap の実装は，Sanders 本人による非常に効率の良いもの[†3]を用いた．

ただし，効率の良い最適経路問合せのためには最短経路問合せには無いいくつかの点を考慮しなければならない．以下これについて説明する．

5.1 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必要がある．このための単純な方法は，経路を各解候補とするものである．しかし，これは多量のメモリを要する．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，最短経路問合せと違い，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が本質的である点．そのため，提案システムでは，どの辺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，通常この計算は軽量であること，頂点に比べ辺を記憶するのは重いことをふまえ，このような実現としている．

5.2 拡大グラフ構築の回避

定理 1 では入力されるグラフ G よりはるかに大きなグラフに対し最短路を計算することを考えている．しかし，この大きなグラフを実際に構成するのは非効率である．提案システムでは，各解候補に対し $state$ の値を覚えることでこの大きなグラフの構築を避けている．同じ G の頂点への経路であっても $state$ の値が違うものはその値ごとに最短路を覚える，というかたちで Dijkstra アルゴリズムを多少変更することでこれは達成される．

5.3 記憶する情報の圧縮

関数 $state$ 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．

†3 www.mpi-inf.mpg.de/~sanders/programs/spq/

しかし，*state* の値域は高々有限である．かつ，*state* の値域の大きさに計算量が依存することを考えると，*state* 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，提案システムの実装では，各解候補の到達頂点と *state* の値をビットフィールドを用いて 1 ワード (すなわち，32 ビットアーキテクチャであれば 4 バイト) で記憶している．これにより，例えば 32 ビットアーキテクチャであれば，拡大後のグラフの頂点が 2^{32} を越えるものは扱えなくなる．しかし，最短経路の計算には一般には少なくともグラフの頂点数に比例するメモリ量を要することをふまえると，これより頂点が多くなる場合を考える必要は無いはずである．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．さらにハッシュ関数を得るにも好都合である．これについては次に述べる．

5.4 到達済み頂点とその距離の記憶

Dijkstra アルゴリズムでは，各頂点が既に到達済みであるか否かを，そして到達済みの頂点に対してはその頂点までの距離を記憶する必要がある．通常，これは頂点をインデックスとする配列で管理される．しかし，最適経路問合せで同様の実装を行おうとすると，G を拡大したグラフの各頂点をインデックスとした配列を用意する必要がある．しかし，先に述べたとおり，拡大したグラフを考えるのは非効率である．

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これらの値を管理している．ハッシュマップを効率よく用いるためには適切なハッシュ関数が必要であるが，拡大したグラフの頂点はすでに 1 ワードで表現されているため，これにマスクをかけることで容易に手に入る．

実装されているハッシュマップの挙動は以下である．まず，ハッシュマップの初期サイズは入力グラフの頂点数とし，ハッシュ関数としては頂点番号に対応する部分にマスクをかけたものとする．ハッシュマップの要素数が増加し再ハッシュが起こる度に，サイズは倍増させ，ハッシュ関数は頂点番号に加えて *state* の値をさらに 1 ビット利用するようにする．これにより，実用上比較的コンフリクトが少なくかつ極めて軽量なハッシュ関数を利用できる．

5.5 まとめ

以上をふまえ，図 2 に提案システムで用いた最適経路問合せアルゴリズムを示す．このアルゴリズムの時間算量は，入力グラフの頂点数と辺数をそれぞれ V および E (ただし $V < E$)，*state* の値域の大きさを k として $\mathrm{O}(kE\log(kE))$ である．

6 実験

提案システムの効率を確認するため boost graph library [23] との比較を行った．Boost graph library は C++におけるグラフアルゴリズムの実装としては事実上の標準である．加えて，boost graph library には，最適経路問合せの一種であるリソース制約付き最短路問合せ [11] のための関数が実装されている．そのため比較対象として適切であろうと判断した．

なお，boost graph library 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いため，提案システムが出力するプログラムに対応するプログラムを人手で記述する必要がある．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及び労力ははるかに大きい．

6.1 実験環境

実験環境は，CPU が dual quad-core Xeon X5550 2.66 GHz，メモリが 12GB，コンパイラが GCC 4.5.0，OS が Linux 2.6.31 (Ubuntu 9.10) 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．

問合せ仕様としては，2 点間最短路 (以下 SP)，3 点間最短路 (指定された頂点を経由する最短路を求めるもの．以下 3SP)，乗換コスト付き最短路 (電車に乗車する際に待ち時間としてコストを要するもの．以下 TRC)，乗換回数制限最短路 (3.2 節で示したもの．以下 TRL) を用いた．比較のため，boost graph library のリソース制約付き最短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．また，SP については Dijkstra アルゴリズムを用いた実

入力: グラフ $G = (V, E)$，関数 $state$，目的関数 f，制約条件 p
出力: 最適経路 r

1. プライオリティキュー Q およびハッシュマップ H を $Q := \emptyset$，$H := \emptyset$ で初期化.
2. 全ての $v \in V$ について，Q に $(f(v), (v, state(v)))$ を insert し，$H[(v, state(v))] := ((), f(v))$ を設定.
3. $Q \neq \emptyset$ の間以下を実行する. なお $Q = \emptyset$ ならば最適経路は存在しない.
   (a) $(w, (v, s)) \leftarrow \mathrm{extract\_min}(Q)$.
   (b) $p(s)$ が真ならば，r を頂点 v のみからなる列とし，ステップ (4) へ進む.
   (c) $H[(v, s)] = ((v', s'), w')$ なる w' が存在しないかまたは $w < w'$ ならば，v を始点とする全ての辺 e_v について以下を行う.
      i. e_v の終点を v_2，$s_2 = state(s \overset{e_v}{\rightsquigarrow} v_2)$，$w_2 = f((v, s) \overset{e_v}{\rightsquigarrow} v_2)$ とする.
      ii. $H[(v_2, s_2)] = ((v_2', s_2'), w_2')$ なる w_2' が存在しないまたは $w_2 < w_2'$ ならば，$H[(v_2, s_2)] := ((v, s), w_2)$ とし，Q に $(w_2, (v_2, s_2))$ を insert.
4. $H[(v, s)] = (vs, w')$ とする.
5. $vs = ()$ ならば r が最適経路. アルゴリズムを終了する.
6. $vs = (v', s')$ とし，始点が v' 終点が v で $state(s' \overset{e}{\rightsquigarrow} v) = s$ かつ $f((w', s') \overset{e}{\rightsquigarrow} v) = w$ なる辺 e を発見する.
7. $v := v'$，$s := s'$ とし，また r は $v' \overset{e}{\rightsquigarrow} v$ を r の先頭に連結した経路とする. ステップ (4) に戻る.

図 2 最適経路問合せアルゴリズム. なお，$state$，f，p は経路ではなく状態 s と目的関数値 w を入力の一部として適切に値を計算できるとする. 例えば，$f(x \overset{e}{\rightsquigarrow} v) = expr(f(x), h_1(x), \ldots, h_n(x), p_1(x), \ldots, p_m(x))$ として $f((w, (s_{h_1}, \ldots, s_{h_n}, s_{p_1}, \ldots, s_{p_m})) \overset{e}{\rightsquigarrow} v) = expr(w, s_{h_1}, \ldots, s_{h_n}, s_{p_1}, \ldots, s_{p_m})$ とする.

装 (BD-SP) も用意した. 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 種類の始点及び終点 (3SP については経由点も) を生成し，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. なお，問合せ時間には入出力に要した時間は含まれず，また使用メモリ量にはグラフの記憶に要したメモリ量が含まれる.

6.2 道路網グラフに対する実験

まず，9th DIMACS implementation challenge: shortest paths [6]†4 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†5. 全ての例について，提案システムは boost graph library の実行速度を大きく上回った.

表 1 アメリカ道路網グラフの大きさ

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

まず，2 点間最短路について見る. 提案ライブラリは，boost graph library での Dijkstra アルゴリズムの実装 (BD-SP) に比べ 2 倍から 3 倍程度早い. これは主に sequence heap を用いたことによる改善である. BD-SP ではプライオリティキューを relaxed heap [8] を用いて実現しており，グラフの頂点数を

†4 www.dis.uniroma1.it/~challenge9/

†5 USA-E および USA-W について，TRL の実行時間が比較的短いことを不思議に思うかもしれない. これは，利用可能な辺をすぐに使い切り解なしとして終了した例が多かったためである.

表 2　アメリカ道路網に対する実験の結果: 平均実行時間 (秒) / 最悪使用メモリ量 (MB)

	NY	FLA	CAL	LKS	USA-E	USA-W
SP	0.06/ 82	0.28/ 341	0.42/ 482	0.49/ 702	0.66/ 902	1.25/1555
BD-SP	0.12/ 78	0.41/ 317	0.85/ 486	1.15/ 698	1.63/ 898	3.34/1553
BR-SP	0.38/102	1.49/ 433	3.58/ 653	4.15/ 959	5.76/1235	10.78/2145
3SP	0.20/110	0.56/ 381	1.02/ 562	1.45/ 789	2.05/1013	4.07/1752
BR-3SP	0.88/130	3.60/ 548	6.42/ 855	9.73/1254	13.51/1619	24.67/2815
TRC	0.10/ 87	0.28/ 350	0.55/ 534	0.67/ 726	0.91/ 957	1.71/1603
BR-TRC	0.66/111	2.56/ 466	3.73/ 712	5.62/1043	7.73/1342	14.49/2333
TRL	6.24/332	6.28/ 483	8.70/ 848	28.28/1755	11.50/1277	6.95/1611
BR-TRL	37.99/774	20.15/1171	24.20/2163	110.22/5008	31.12/3190	17.72/3396

V，辺数を E として，時間計算量は $O(V \log V + E)$ である．これは提案システムのものより良い．しかし，sequence heap は relaxed heap に比べキャッシュに乗りやすいため，総合的に見て高速な処理を実現している．また，提案システムは BD-SP に比べやや多くのメモリを消費する．これは，元にしたアルゴリズムが Dijkstra アルゴリズムの decrease-key を用いない変種であるためである．このアルゴリズムでは，既に発見済みであるが最短路は未確定である頂点を再発見した際に decrease-key の代わりに insert を行うため，通常のアルゴリズムに比べメモリを多く消費する．なお，boost graph library のリソース制限付き最短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．ちなみにリソース制限付き最短路アルゴリズムは Dijkstra アルゴリズムに似たアルゴリズムを用いており，プライオリティキューの実装には二分ヒープを用いている．時間計算量は $O(E \log V)$ である．

それ以外の例についても，一貫して提案システムの実装は boost graph library のリソース制限付き最短路アルゴリズムによるものに比べてかなり高速であった．より特筆すべきは，特に TRL において消費メモリ量に大きな差が出ている点である．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，boost graph library 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．しかし，これは非常に煩雑であり，また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．

6.3　ランダム生成されたグラフに対する実験

次に，各辺の両端点及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 8192 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間の辺数で数えた平均距離が近く，また今回のものは辺重みの分散も大きいため，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．そのため，Dijkstra アルゴリズムに基づく最短路問合せでは比較的扱い辛い入力となっている．

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 4 に示す．やはり，提案システムは一貫して boost graph library を用いた実現より高速に問合せを行えている．ただし，時間計算量から予想されるとおり，辺数の多いグラフでは提案システムと boost graph library での Dijkstra アルゴリズムの実装の速度差は小さい．なお，LARGE に対する TRL

表 4 ランダム生成グラフに対する実験の結果: 平均実行時間 (秒) / 最悪使用メモリ量 (MB)

	SMALL	MIDDLE	SPARSE	DENSE	LARGE
SP	0.44/ 716	0.94/1442	0.52/ 668	1.13/1955	1.46/2644
BD-SP	0.66/ 274	1.55/ 546	1.55/ 451	1.24/ 994	2.48/1090
BR-SP	6.46/ 486	14.39/ 971	5.34/ 633	33.32/2052	32.84/1941
3SP	1.01/ 751	2.12/1473	1.58/ 674	3.37/2185	4.78/2814
BR-3SP	10.10/ 544	22.26/1084	7.70/ 812	55.24/2758	51.07/2165
TRC	0.47/ 863	1.23/1589	0.79/ 714	1.63/2244	2.02/3077
BR-TRC	7.76/ 565	17.06/1098	21.01/1266	41.83/2463	39.64/2194
TRL	1.95/1796	4.85/3586	5.34/ 669	5.87/5063	
BR-TRL	20.78/ 947	49.03/2014	13.43/1552	132.43/4857	

表 3 ランダム生成グラフの大きさ

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

は消費メモリが多すぎて実行できなかった.

今回の例では，道路網の場合とは異なり，提案システムはかなり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，decrease-key の代わりに insert を行うケースが多数発生していると想像される．グラフごとに適切なアルゴリズムを選べるよう，アルゴリズムをパラメータとしてとれるような実装を行うことには意義があると思われる.

7 まとめと今後の課題

本発表では，最適経路問合せを効率よく行うシステムの実装とその実験結果を示した．丁寧な設計及び実装により，提案システムは単純な仕様記述から効率の良い問合せを実現できていることが確認できた.

今後の研究としては，同様のアイデアでコンパイラのコード最適化を記述する可能性を考えている．正規経路問合せによってコンパイラのコード最適化を記述することで，最適化器を系統的に得る試み [5, 14] がなされているが，提案手法は単に正規言語制約を満たす部分構造を取り出すだけでなく，その中から何らかの基準で最適なものを選ぶこともできる．これを用いて，発見的なコード最適化も系統的な記述から導出できないかと考えている.

また，定理 1 は有限状態オートマトン同士の積構成に基づいているが，同様の着想は文脈自由文法と有限状態オートマトンの組合せへと拡張できる．すなわち，例えば各辺にトークンの割り当てられたグラフの経路から，そのトークン列を式として評価した場合に最大となる経路を発見する問題や，文脈自由文法で特徴づけられるシステムの実行トレースの中で最適なものを求めることなどは，同様にして解くことができる．このような一般化，そのための領域限定言語の設計，その効率の良い実現等は興味深い研究課題だと考えている.

謝辞 実験環境を提供してくださった東京大学情報理工学系研究科数理情報学専攻の数理第七研究室に感謝する.

参 考 文 献

[1] 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 - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*, Lecture Notes in Computer Science, Vol. 2461, Springer,

2002, pp. 126–138.

[2] Barrett, C. L., Jacob, R., and Marathe, M. V.: Formal-Language-Constrained Path Problems, *SIAM Journal on Computing*, Vol. 30, No. 3(2000), pp. 809–837.

[3] Chan, E. P. F. and Zhang, J.: A fast unified optimal route query evaluation algorithm, *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, CIKM 2007, Lisbon, Portugal, November 6-10, 2007*, ACM, 2007, pp. 371–380.

[4] Chowdhury, R. A.: *Cache-efficient Algorithms and Data Structures: Theory and Experimental Evaluation*, PhD Thesis, Department of Computer Sciences, Univerity of Austin, 2007.

[5] de Moor, O., Lacey, D., and Wyk, E. V.: Universal Regular Path Queries, *Higher-Order and Symbolic Computation*, Vol. 16, No. 1–2(2003), pp. 15–35.

[6] Demetrescu, C., Goldberg, A. V., and Johnson, D. S.(eds.): *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, American Mathematical Society, 2009.

[7] Desrochers, M. and Soumis, F.: A Generalized permanent labeling algorithm for the shortest path problem with time windows, *INFOR*, Vol. 26(1988), pp. 191–212.

[8] Driscoll, J. R., Gabow, H. N., Shrairman, R., and Tarjan, R. E.: Relaxed Heaps: An Alternative to Fibonacci Heaps with Applications to Parallel Computation, *Communcation of the ACM*, Vol. 31, No. 11(1988), pp. 1343–1354.

[9] Flesca, S., Furfaro, F., and Greco, S.: Weighted path queries on semistructured databases., *Information and Computation*, Vol. 204, No. 5(2006), pp. 679–696.

[10] Ioachim, I., Gélinas, S., Soumis, F., and Desrosiers, J.: A dynamic programming algorithm for the shortest path problem with time windows and linear node costs, *Networks*, Vol. 31, No. 3(1998), pp. 193–204.

[11] Irnich, S. and Desaulniers, G.: Shortest Path Problems with Resource Constraints, *Column Generation*, Desaulniers, G., Desrosiers, J., and Solomon, M. M.(eds.), Springer, 2005, chapter 2, pp. 33–65.

[12] Joksch, H. C.: The shortest route problem with constraints, *Journal of Mathematical Analysis and Applications*, Vol. 14, No. 2(1966), pp. 191–197.

[13] Korkmaz, T. and Krunz, M.: Multi-Constrained Optimal Path Selection, *Proceedings IEEE INFOCOM 2001, The Conference on Computer Communications, Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies 22-26 April 2001, Anchorage, Alaska, USA*, IEEE, 2001, pp. 834–843.

[14] Liu, Y. A., Rothamel, T., Yu, F., Stoller, S. D., and Hu, N.: Parametric regular path queries., *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation 2004, Washington, DC, USA, June 9-11, 2004*, ACM, 2004, pp. 219–230.

[15] Mendelzon, A. O. and Wood, P. T.: Finding Regular Simple Paths in Graph Databases., *SIAM Jornal on Computing*, Vol. 24, No. 6(1995), pp. 1235–1258.

[16] Morihata, A., Matsuzaki, K., and Takeichi, M.: Write it Recursively: A Generic Framework for Optimal Path Queries, *Proceedings of the 2008 ACM SIGPLAN International Conference on Functional Programming, ICFP 2008, Sept. 22-24, 2008, Victoria, BC, Canada*, ACM, 2008, pp. 169–178.

[17] Orda, A. and Rom, R.: Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Length, *Journal of the ACM*, Vol. 37, No. 3(1990), pp. 607–625.

[18] Punnen, A. P.: A linear time algorithm for the maximum capacity path problem, *European Journal of Operational Research*, Vol. 53, No. 3(1991), pp. 402–404.

[19] Ramachandran, V.: Cache-Oblivious Computation: Algorithms and Experimental Evaluation, *2007 International Conference on Computing: Theory and Applications (ICCTA 2007), 5-7 March 2007, Kolkata, India*, IEEE Computer Society, 2007, pp. 20–26.

[20] Romeuf, J.-F.: Shortest Path Under Rational Constraint, *Information Processing Letters*, Vol. 28, No. 5(1988), pp. 245–248.

[21] Sanders, P.: Fast Priority Queues for Cached Memory, *ACM Journal of Experimental Algorithmics*, Vol. 5(2000), pp. 7.

[22] Sherali, H. D., Jeenanunta, C., and Hobeika, A. G.: The approach-dependent, time-dependent, label-constrained shortest path problem, *Networks*, Vol. 48, No. 2(2006), pp. 57–67.

[23] Siek, J. G., Lee, L.-Q., and Lumsdaine, A.: *The Boost Graph Library: User Guide and Reference Manual*, Addison-Wesley, 2001.

[24] Villeneuve, D. and Desaulniers, G.: The shortest path problem with forbidden paths, *European Journal of Operational Research*, Vol. 165, No. 1(2005), pp. 97–107.