最短ツリーの記憶による交通量配分の経路探索の効率化*

Acceleration of Shortest-Path Search in Traffic Assignment by Memorizing Shortest-Path Trees*

鷹尾和享**・朝倉康夫***
By Kazutaka TAKAO**・Yasuo ASAKURA***

１．はじめに

交通量配分には種々のアルゴリズムがあるが、その多くは多数回の繰り返し計算を必要とする。その過程で最短経路探索が何度も行われるため、配分計算全体のパフォーマンスは最短経路探索の実装方法に強く影響される。実務においては一般にネットワークのサイズが大きく、長い計算時間を要する。そのうえ、たとえば、高速道路のハーフランプや、リンクコストの足し算では表現できない(non-link-additive)料金体系のような、実際のネットワークの様子を正確に表現することがしばしば要求され、そのためにネットワーク表現が複雑になり、それが配分計算をさらに重くしている。

本稿は、前回の iteration での最短ツリーを記憶することで、最短経路探索を高速化する方法について述べる。ラベル修正法は最短経路探索のアルゴリズムの一つであり、探索の過程でノードに付いた長経路のラベルを、より短い経路のラベルに修正していくことで、ある根ノードからの最短ツリーを得る。記憶しておいた前回の最短ツリーを利用し、各ノードのコストを今回の iteration 用に更新すると、その値は、長経路のラベルが付くのを阻止するための足切りラインとして用いることができ、ラベル修正の回数を減らすことができる。

これまで、配分計算を高速化する種々の試みが行われてきたが、既存の報告は、繰り返し回数が少ない場合や、小さなネットワークでの計算例しか報告していない場合が多い。しかし、実務では、配分計算をできるだけ短時間で終わらせることは切迫した問題である。したがって、本稿では、実際のネットワーク上で実際的な繰り返し回数で提案手法を適用した場合のパフォーマンスを調べることを重視する。すなわち、本稿の目的は、最短ツリーの記憶法を実際的なネットワーク表現に適用し、そのパフォーマンスを報告することである。今日では、PC のメモリサイズが大きくなり、1GB 規模のメモリを確保することも難しくない。したがって、豊富なメモリをいかに上手に使うかを議論する良い時期である。

提案手法は Takao and Asakura (2007)[1]でも速報したが、本稿では無駄な消費メモリを抑えた最新の計算結果について報告する。

* キーワード：交通ネットワーク分析、経路選択
** 正員、博士（工）
（社）システム科学研究所　（〒 604-8223　京都市中京区新町通四条上ル小結棚町 428　新町アイエスビル　TEL: 075-221-3022, FAX: 075-231-4404）
*** 正員、工博
神戸大学大学院　工学研究科　教授

２．関連研究

Frank-Wolfe(FW)法は、大規模なネットワークに簡単に適用できることから、広く用いられている配分手法である。しかし、FW 法は一般に収束が遅く、多数回の繰り返しを必要とする。たとえば、Boyce *et al.* (2004)[2]は 2000 回という値を挙げている。そこで、FW 法を改良するいくつかの方法が開発されている [3]。たとえば、補助解のフローパターンを改良する方法(Lee and Nie, 2001)[4]や、ステップサイズを改良する方法(Weintraub *et al.*, 1985)[5]があり、また、別の新しいアルゴリズムを用いる方法(たとえば Dial, 2006)[6]もある。

FW 法の実行過程では最短経路探索が何度も行われる。これまでに多くの最短経路探索のアルゴリズムが開発されているが、その多くは最短経路探索の問題をおのおのの探索別に単独で取り扱っており、iteration 間で組み合わせることに着目しているものはあまりない。

本稿で述べる方法に似た方法を Dial (2006)[7]が提案しており、また、Mahmassani *et al.* (1993)[8]も短く触れている。それは、マルチクラスの最短経路探索において、あるクラスの最短ツリーを用いて他のクラスの最短経路探索を高速化するものである。この方法では、クラス間で走行コストが異なるリンクは、全体のうちのごくわずかしかない点に着目している。そこで、あるクラスの最短ツリーが既に求まっている場合、最短経路探索にラベル修正法を用いて別のクラスの最短ツリーを求めると、そのツリーに対してリンクコストの異なる部分だけを補正すればよく、ラベル修正が必要な部分は限られた範囲になる。これに対して、本稿で扱おうとする問題は、シングルクラスの iteration 間での問題である。FW 法が収束していないうちはステップサイズは 0 ではないため、ほとんどのリンクのコストが前回の iteration と異なる

点が彼らの問題とは大きく異なっており、彼らの方法では、ほぼ全てのノードのラベル修正が必要になる。しかし、この、ツリーの記憶というアイデアは、別の方法でiteration 間での最短経路探索の高速化に応用できる。

３．ラベル修正法とツリーの記憶

ラベル修正法は効率のよい最短経路探索アルゴリズムである。探索の間、ノードｎにはラベルと先行ポインタが付いている。ラベルとは、現在探索されているうちで最短の、根ノードからの所要コストであり、先行ポインタとはそれに対応する経路を表す。「スキャン」操作とは、新しく見つかった経路が現在のラベルより低コストかどうかを調べる操作であり、ｎの下流ノードmの現在のラベルと、新しく見つかったｎからの経路のコストを比較し、新しい方が低コストであれば、mのラベルを修正し、mをキュー（下流ノードをスキャンする必要のあるノードのリスト）に加える(enqueue)。この操作を繰り返すことで最短ツリーを求める。

この時、mが既にキューから取り出され(dequeue)ていた場合、mは再度キューに加わることになる。このような場合、mの下流ノードが連鎖的にキューに再加入することになり、その多くがラベル修正を引き起こし、それがさらにキューへの再加入を引き起こす。したがって、最短でないラベルが一時的に付いてしまうと、非効率的である。

そこで、前回の iteration の最短ツリーを記憶しておくことで、この現象を緩和することができる。リンク走行コストは前回の iteration と異なるため、各ノードの根ノードからの所要コストは、今回の iteration 用に更新しなければならない。その結果、記憶したツリーは最短ツリーではなくなる可能性がある。しかし、最短ではないにせよ、そのコストの経路が存在すると解釈することができる。したがって、更新後の所要コストを足切りラインとして利用し、それより大きいコストのラベルが付くのを阻止することができる。

手順をまとめると次のようになる。

(1) 初回は普通に最短経路探索を行う。ただし、求まったツリーは記憶する。
(2) 次回以降は足切りラインを利用しながらラベル修正法を行う。全ての根について次の操作を行う。
 (2-1) ツリーの所要コストを更新する。これが足切りラインとなる。
 (2-2) もしスキャンされたコストが足切りラインよりも大きければ、無視する。そうでなければ、通常通りのラベル修正法を行う。

足切りラインを利用する段階で、所要時間が全く同じ経路が複数存在した場合、スキャンの順序が通常のラベル修正法と異なる場合がある。その結果、たとえば FW 法の場合、収束が不十分だと、配分結果のリンクフローは異なる場合がある。

FW 法に関して言えば、繰り返しの初期の頃は一次元探索のステップサイズが大きい傾向がある。そのため、記憶したツリーの更新後の所要コストは、現在の iteration の最短コストと比べてかなり大きくなる傾向があると言える。その場合、足切りラインが効率よく働かない可能性がある。しかし、繰り返しを重ねるにつれてステップサイズは小さくなっていくので、足切りラインは効率的に働くようになり、計算時間は劇的に改善していくと期待できる。

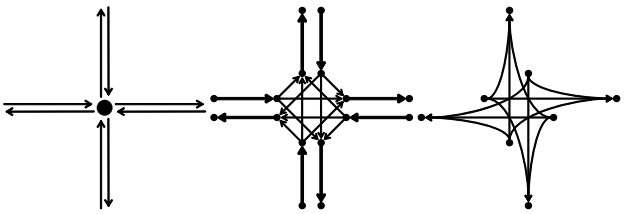
一方、ツリーの記憶のために追加的に必要となるメモリは次のように見積もることができる。

追加メモリ量 ＝ 先行ポインタのバイト数×
ノード数×セントロイド数

４．実際的なネットワーク表現

（１）交差点展開

現実の道路網を正確にモデル化したい場合、ネットワーク表現も複雑になり、最短経路探索も重くなる。



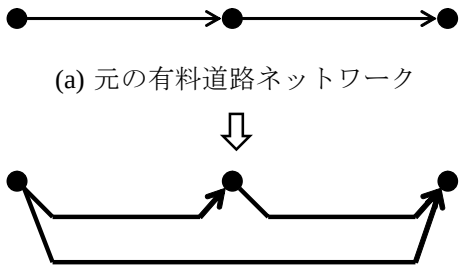
(a)シングルノード　(b)交差点展開Ⅰ　(c)交差点展開Ⅱ

図－１　交差点展開

現実の道路網では、片方向しか出入りできないようなランプ（ハーフランプ）や、右左折制限のある交差点が存在する。このような様子を正確に表現したい場合、図－１(a)のようなシングルノードの形では最短ツリーが単純なツリーにならない場合がある。なぜなら、あるノードに関して、流出方向毎に先行ポインタが異なる場合が起こりうるためである。そこで、(b)のように交差点展開をすれば、この問題を解決することができ、普通に経路探索を行うだけで最短ツリーを正しく求めることができるようになる。さらに、(c)のように、より簡略化された形に変形することもできる。この交差点展開は最短経路の探索時にのみ必要であり、自動的に展開形状に変換することができる(Takao *et al.*, 2006)[9]。

（２）Entry-Exit モデル

現実の有料道路は link-additive ではない料金体系になっている場合がある。たとえば、長距離の利用区間の料金が幾分逓減されている場合がある。その場合、長距離トリップの場合は極力有料道路を走り続ける方が最短コストになることがある。そのような場合、最短ツリーは単純なツリーにならない場合があり、通常の方法では長距離の利用区間の料金や最短経路をうまく表現できない。そこで、Yang *et al.* (2004)[10]は、各利用区間を仮想リンクに置き換える Entry-Exit モデルを提案した（図－２）。本稿の Entry-Exit モデルでは、主に阪神高速の乗り継ぎ料金を表現するため、阪神高速のリンクを仮想リンクに置き換える。Yang *et al.* (2004)の元々の Entry-Exit モデルは入口と出口で料金が定まる（entry-exit based）ことを前提としているが、経路によって料金が異なる場合（path based)、入口－出口間を複数本の仮想リンクで結ぶことで、簡単に応用することができる(Takao and Higashi, 2006)[11]。



(a) 元の有料道路ネットワーク

(b) Entry と Exit の組を仮想リンクで接続

図－２　Entry-Exit モデル

５．数値計算例

（１）テストしたモデル

提案手法の性能を見るため、実際のネットワークを用いて配分計算を行った。次の３つのネットワーク表現モデルを用いて評価を行った。それぞれについて、提案手法のツリー記憶法と、通常のラベル修正法とで FW 法の利用者均衡配分を行って性能を比較した。

- Link-Additive モデル … 交差点展開Ⅱ, link-additive 料金
- Entry-Exit モデル … 交差点展開Ⅱ, Entry-Exit モデル料金
- 単純モデル … シングルノード, link-additive 料金

単純モデルは各ノードの流入リンクと流出リンクの全ての組み合わせは通行可能とするモデルである。

用いたネットワークは近畿地方全域のネットワークであり、その規模は表－１に示す通りである。セントロイド数は 1140 である。OD ペア数はそれら相互間だが、１つの O ごとに全 D への最短経路を一度に探索するので、性能に主に影響するのはセントロイド数である。

FW 法の繰り返し回数は 100 回とした。初期実行可能解が１回分あるので、セントロイドあたりの最短経路探索の回数は 101 回である。キューの実装は double ended queue、すなわち、初加入ならば末尾に、再加入ならば先頭に追加する方法を採用した。Pentium 4 (630) 3GHz の PC を使用し、プログラムは Visual C++ 6.0 でコンパイルし、Windows XP 上で実行した。

なお、プログラムの作り方の差異による性能への影響を軽減するため、ツリーを記憶する場合としない場合とで同一の実行モジュールを使用し、パラメータを読み込んで両者を切り替える方法で実行した。また、作業領域は必要サイズ分を動的に取得するようにし、巨大なメモリを固定的に確保しすぎないように工夫した。このほか、データ構造の選択やアルゴリズムの実装方法も全体の性能に少なからず影響すると考えられるが、この論文の範囲ですべてを述べることは難しいので、これについては今後の課題としたい。

表－１　ネットワーク規模

指標	モデル		
	Link-Additive	Entry-Exit	単純
ノード数(元 network)	10850		
ノード数(経路探索)	31680	31465	10850
リンク数(元 network)	15328		
仮想リンク数	0	34114	0
リンク数(経路探索)	67218	75211	30529

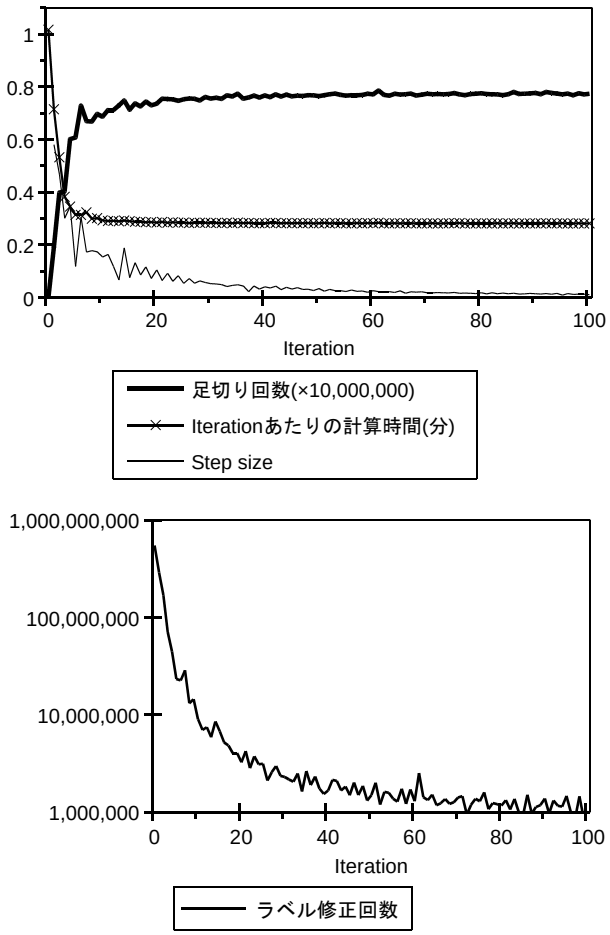
（２）Link-Additive モデルの結果

表－２に Link-Additive モデルの結果を示す。計算時間は繰り返し 100 回の FW 法のトータルの CPU 時間を Windows API により取得したものであり、データ入出力の時間を含んでいる。Max.メモリとは、実行中の最大消費メモリ（プログラム領域を含む）を Windows API で調べたものである。「通常の」とは、ツリーを記憶しない従来の方法である。比較のため、Dijkstra 法（ラベル確定法）による結果も示した。通常のラベル修正法でも Dijkstra 法より高速であるが、提案手法はそれよりもさらに大幅に高速である（104 分→ 30 分)。

表－２　Link-Additive モデルの結果

用いた最短経路探索	計算時間	Max.メモリ
通常の Dijkstra 法	197.6 分	25.0 MB
通常のラベル修正法	104.4 分	25.0 MB
ツリー記憶ラベル修正法	30.3 分	166.5 MB

※繰り返し 100 回



図－３　ラベル修正回数・関連指標の推移
（Link-Additive モデル）

表－３ ラベル修正の内訳回数（Link-Additive モデル）

	通常	ツリー記憶
足切り	0	7,721,977
ラベル修正,queue 再加入	533,398,211	767,187
queue 内でラベル修正	51,390,646	532,578
スキャンのみ	599,297,270	29,488,624

※後半の iteration 1 回あたり

これは、ラベル修正回数が劇的に減少するからである。繰り返しが進むにつれてステップサイズが小さくなり、足切りが効率的に働くようになり、ラベル修正回数の減少に寄与する（図－３）。この、高速化が実現できた子細を分析・確認するため、高速化の効果が特に大きい、後半の 51 ～ 100 回目について、ラベル修正プロセスの内訳の回数を調べ、表－３に示した。これは iteration 1 回あたりの平均回数である。

ラベル修正動作は２つのパターンに分けることができる。１つは既にキューから取り出された後のノードについてのラベル修正であり、もう１つはまだキューに入っているノードについてのラベル修正である。前者は、当該ノード自身のラベル修正のみならず、下流ノードのスキャン・ラベル修正・キューへの再加入を引き起こすため、ツリー探索全体のパフォーマンスへの影響が大きいので、注視する必要がある。

足切りの結果、ラベル修正によるキューへの再加入の回数が 5 億回余から 76 万回余へと劇的に減少している。また、その結果、ラベル修正回数のみならず、スキャンの回数の減少にもつながっている。

その一方で、最大消費メモリはかなり増加することになるが、166.5MB は最近の PC で十分実行可能なサイズである。

（３）Entry-Exit モデルの結果

Entry-Exit モデルの結果を表－４に示す。ツリー記憶法は通常のラベル修正法より高速となった。なお、仮想リンクの効果で通常のラベル修正法でも幾分速くなっており、ツリー記憶の効果は幾分小さくなっている。

つまり、仮想リンクは遠く離れたノードを直接１本で結ぶ場合があるが、その経路は高速道路を経由するものであるため、長距離トリップの場合はそれが最短経路を構成する場合が多い。したがって、通常のラベル修正法でも、最終的に最短となるラベルが経路探索の早い段階で付く傾向があることを期待できる。このことは、表－５の通常のラベル修正法の各回数が表－３よりも幾分小さくなっていることからもわかる。一方、出口ノード１つにつき全入口ノードからのスキャンが行われるため、ツリー記憶法のスキャンのみの回数（表－５）は Link-Additive モデル（表－３）と比べて若干増えており、計算時間もわずかに長くなっている。

表－４　Entry-Exit モデルの結果

用いた最短経路探索	計算時間	Max.メモリ
通常の Dijkstra 法	219.5 分	72.8 MB
通常のラベル修正法	54.9 分	72.8 MB
ツリー記憶ラベル修正法	32.0 分	217.3 MB

※繰り返し 100 回

表－５ ラベル修正の内訳回数（Entry-Exit モデル）

	通常	ツリー記憶
足切り	0	9,582,091
ラベル修正,queue 再加入	154,368,094	812,788
queue 内でラベル修正	37,997,169	529,288
スキャンのみ	290,220,571	37,002,302

※後半の iteration 1 回あたり

（４）単純モデルの結果

単純モデルの結果を表－６・表－７に示す。単純モデルでもツリー記憶法の効果は見られ、通常のラベル修正

法よりかなり高速になった。ラベル修正の回数も他のモデルと同様、劇的に減少している。

表－６　単純モデルの結果

用いた最短経路探索	計算時間	Max.メモリ
通常の Dijkstra 法	29.3 分	17.0 MB
通常のラベル修正法	32.1 分	17.0 MB
ツリー記憶ラベル修正法	12.8 分	65.5 MB

※繰り返し 100 回

表－７　ラベル修正の内訳回数（単純モデル）

	通常	ツリー記憶
足切り	0	3,141,389
ラベル修正,queue 再加入	139,389,155	363,787
queue 内でラベル修正	16,817,378	262,980
スキャンのみ	274,379,448	19,796,808

※後半の iteration 1 回あたり

（５）２回前のツリーの利用

これまでのツリー記憶法で用いた最短ツリーは直前の iteration のツリーである。FW 法の収束が進むにつれて複数の経路が均衡状態に近づくため、たとえば、２経路による均衡状態になる場合には、その２経路間で iteration ごとに最短経路が交互に切り替わる場合もある。ネットワーク全体のうちで、そのようなケースが特に多い場合には、直前ではなく、２回前の iteration の最短ツリーを用いる方が、今回の最短経路になっている確率が高く、足切りがより効果的に働く可能性がある。ただし、その反面、２回分のツリーを記憶することになるので、その分メモリ消費量は増大する。

表－８に直前の iteration のツリーを用いた場合と、２回前の iteration のツリーを用いた場合の結果を比較したものを示す。パフォーマンスはマシン環境（キャッシュメモリのサイズ等）によっても異なってくると思われるが、筆者らの環境では２回前のツリーを用いる方がわずかに高速になっている。最大消費メモリはかなり大きくなっており、わずかな速度向上を取るか、メモリの少なさを取るかのトレードオフとなる結果が出た。

表－８　２回前のツリー利用の結果比較

モデル, 用いたツリー	計算時間	Max.メモリ
Link-Additive, 直前	30.3 分	166.5 MB
Link-Additive, ２回前	30.2 分	308.8 MB
Entry-Exit, 直前	32.0 分	217.3 MB
Entry-Exit, ２回前	31.8 分	362.3 MB
単純, 直前	12.8 分	65.5 MB
単純, ２回前	12.7 分	114.0 MB

※繰り返し 100 回

６．考察

（１）高速道路の存在による効果の違い

ツリー記憶の効果はネットワーク形状によって異なると考えられる。高速道路が存在する場合、高速道路の長いリンクは遠方のノードを直接結ぶため、通常のラベル修正法では根から遠いノードが比較的早い時点でキューに入る傾向がある。しかし、そのラベルが最短ではなかった場合、ラベル修正とキューへの再加入を引き起こす原因となる。このため、大きなコストのラベルの足切りをすることでこの問題を大幅に緩和することができる。したがって、高速道路を持つネットワークは提案手法の効果が大きいと思われる。

（２）Dijkstra 法との組み合わせ

また、提案手法は一見 Dijkstra 法にも適用可能なように思われるが、高速化の効果は期待できない。その理由は、Dijkstra 法の「仮ラベルの付いたノードの中から最小コストのノードを選ぶ」という操作が、大きなコストのラベルが付くのを既に防いでいるからである。

特に、交差点展開で交差点コストを無視する場合、効果は全くない。図－１(b)の例では、１個の流出ノードは３本の交差点内リンク（図の細線）によって３個の流入ノードと結ばれている。この３本のリンクのリンクコストは等しい、すなわち 0 なので、「３つの流入ノードから最小コストのノードを選ぶ」という操作が行われることで、その先の流出ノードには最終的な最短ラベルが最初に付くことになり、一時的に大きなコストのラベルが付くことは決してない。さらに、その先の、隣の交差点の流入ノードについても、そこに至るリンクは１本しか存在せず、最短ラベルが必ず最初に付くことになる。その結果、ツリー記憶による足切りの効果は全くない。

また、シングルノード表現の場合についても、足切りがなされる場合は限られ、また、たとえ足切りがなされても、最小コストのノードを選ぶ操作をわずかに軽くするだけである。むしろ、ツリーのコストの更新のオーバーヘッドの方が高く付くことが多い。

したがって、提案手法との組み合わせによって、Dijkstra 法よりもラベル修正法の方が速度の面でより有利になる傾向があると言える。

７．結論

本稿では、最短ツリーを記憶することでラベル修正法の高速化を図る方法を紹介し、その効果を実ネットワーク上で確認した。本稿の成果は次のように要約される。

(a) FW 法をラベル修正法で組む場合、前回の iteration

の最短ツリーを記憶することで高速化が図れる。

(b) 記憶したツリーは、長距離のラベルが付くのを防ぐための足切りラインとして利用できる。

(c) FW 法の計算時間の改善効果の有無や度合いはネットワークデータや実行環境によって異なると考えられるが、本稿の計算例のように大幅に高速化できる場合があることがわかった。

本稿では FW 法について述べたが、分割配分法も iteration ごとに最短経路探索を繰り返すため、同様に本手法の適用効果がある。ただし、本稿の手法は繰り返し回数が多い問題に対して効果を発揮する一方、繰り返し回数が少ない場合は計算効率が高くなるとは限らない。たとえば、分割配分法では、5 分割程度では大してメリットは得られないと思われる。

なお、本稿の近畿圏の計算例では PC 上で十分実行できるサイズであったが、ネットワーク規模が特に大きい場合、32bit のメモリ空間を使い果たす恐れがあることに注意する必要がある。

また、それに関連していえば、最近、FW 法と転換率法を組み合わせる配分方法の採用が検討されているが、転換率配分法も別の意味において経路の記憶を必要とする方法であるため、実際的な制約により、本稿の提案手法とは両立できない恐れがある。

参考文献

1) Takao, K. and Asakura, Y.: Acceleration of traffic assignment with practical network representations by shortest-path tree memorization, in Proceedings of the 12th HKSTS, Hong Kong, pp. 165-174, 2007.
2) Boyce, D., Ralevic-Dekic, B., and Bar-Gera, H.: Convergence of traffic assignments: How much is enough?, ASCE Journal of Transportation Engineering, 130(1), pp. 49-55, 2004.
3) 土木学会土木計画学研究委員会交通需要予測技術検討小委員会編: 道路交通需要予測の理論と適用 第Ⅱ編 利用者均衡配分モデルの展開, 第５章, 土木学会/丸善, 2006.
4) Lee, D.H. and Nie, Y.: Accelerating strategies and computational studies of the Frank-Wolfe algorithm for the traffic assignment problem, Transportation Research Record, 1771, pp. 97-105, 2001.
5) Weintraub, A., Ortiz, C., and Gonzalez, J.: Accelerating convergence of the Frank-Wolfe algorithm, Transportation Research Part B, 19(2), pp. 113-122, 1985.
6) Dial, R.B.: A path-based user-equilibrium traffic assignment algorithm that obviates path storage and enumeration, Transportation Research Part B, 40(10), pp. 917-936, 2006.
7) Dial, R.B.: An efficient algorithm for building min-path trees for all origins in a multi-class network, Transportation Research Part B, 40(10), pp. 851-856, 2006.
8) Mahmassani, H.S., Hu, T.Y., Peeta, S., and Ziliaskopoulos, A.: Development and testing of dynamic traffic assignment and simulation procedures for ATIS/ATMS applications, Technical Report DTFH61-90-R-00074-FG, http://www.itsdocs.fhwa.dot.gov//JPODOCS/REPTS_TE//7884.pdf, 1993.
9) Takao, K., Higashi, T., Yasuda, K., and Asakura, Y.: Network representation and shortest path reflecting the ramp entry or exit direction limitation, in Proceedings of the 13th World Congress on ITS, London, U.K., 1787.pdf, 2006.
10) Yang, H., Zhang, X., and Meng, Q.: Modeling private highways in networks with entry-exit based toll charges, Transportation Research Part B, 38(3), pp. 191-213, 2004.
11) Takao, K. and Higashi, T.: Traffic assignment in a real network with non-link-additive toll charged expressway, in Proceedings of the 11th HKSTS, Hong Kong, pp. 411-420, 2006.

最短ツリーの記憶による交通量配分の経路探索の効率化*

鷹尾和享**・朝倉康夫***

Frank-Wolfe 法の繰り返しループの間、最短経路探索が何度も繰り返し実行されるが、本稿は、前回の最短ツリーを記憶することによって最短経路探索を高速化する方法について述べる。記憶しておいた最短ツリーのリンクコストは今回の繰り返し回用に更新する必要があるが、ラベル修正法において、大きなコストのラベルが付くのを阻止するための足切りラインとして利用できる。本稿ではこの方法を実際的な表現の実ネットワークに適用し、大幅に高速化することを確認した。この理由は、ラベル修正回数が劇的に減少するからである。その一方、必要メモリは増大するが、通常のネットワーク規模では最近のＰＣ上で実行できる程度に収まる。

Acceleration of Shortest-Path Search in Traffic Assignment by Memorizing Shortest-Path Trees*

By Kazutaka TAKAO** · Yasuo ASAKURA***

As the Frank-Wolfe algorithm repeats many iterations, it executes many repetitions of the shortest-path search. This paper describes acceleration of the shortest-path search by memorizing the shortest-path trees of previous iterations. After updating the travel times of the memorized trees for the current iteration, they are used as rejection borderlines to avoid setting long-routed labels in the label-correcting algorithm, contributing to reducing label correction. The performance is considerably improved because the counts of label correction are dramatically reduced. Although the required memory size becomes large, it is within the execution ability of recent PCs.