前処理付き最短経路探索手法の pgrouting への実装

M2009MM029 和久田崇
指導教員：河野浩之

1 はじめに

最短経路探索の古典的な手法として，ダイクストラ法や A*があり，インターネットによる経路探索サービス (WEB ルーティングサービス) などで広く利用されている．しかし，これらの手法は用いる地図が大規模であるほど探索に時間が掛かるという欠点がある．一方近年では道路における最短経路探索の手法として，前処理によってネットワークに補助データを付加する事で探索を高速化する手法が研究されている．しかしこのような手法は実際の GIS ソフトウェアにはまだ実装されていない．

本論文では代表的な前処理を用いた探索手法として Landmark A*(ALT)[1]，Highway Hierarchies(HH)[5]，Arc Flags(AF)[4]，Transit-Node Routing(TNR)[3] を調査し，探索速度が最も高速な手法として [6] で提案されている Transit Nodes と Arc Flags を組み合わせた手法を選択し，オープンソースの経路探索エンジン「pgrouting」に追加することで実装する．また，オープンソースの GIS プラットホーム「Mapserver」を用いて地図や経路探索結果を可視化する．実験では実装した手法と，pgrouting のダイクストラ法や A*を同一の地図データで実行し，前処理時間，探索時間を計測し，比較を行う．最後に実験結果を踏まえ，WEB ルーティングサービスにおいて提案手法が従来の手法より有効であることを確かめる．

2 前処理付き最短経路探索手法の調査

参考文献 [6] では前処理を用いた最短経路手法に関する研究が行われた．以下に，西ヨーロッパ (約 1800 万ノード，4230 万エッジ) の道路ネットワークにおいて行われた，前処理の時間と消費スペース，探索の訪問ノード数と時間を比較する実験の結果を引用する．

表 1 先行研究における実験結果 (西ヨーロッパ)

	前処理		探索	
	時間 [min]	スペース [B/node]	訪問ノード数	時間 [ms]
ALT	13	70	82348	160.3
AF	2156	25	1593	1.1
HH	13	48	709	0.61
TNR	112	204	N/A	0.0034
TNR+AF	229	321	N/A	0.0019

以上の結果から，TNR+AF は前処理に時間が掛かるものの，探索速度においては最も高速な手法であることがわかる．本研究では WEB ルーティングサービスのように 1 度前処理が行われた地図データに対して，不特定多数から探索が行われる場合を考えるため，探索速度を重点に置き，TNR+AF に着目する．

3 TNR+AF の pgrouting への実装

TNR+AF は，ある程度長い最短経路が必ず通るノード (Transit Node) を利用する TNR と，地図を領域分割し，各エッジにどの領域に向かう最短経路に用いられるかを示す flag vector を設定する AF という手法を組み合わせたものである．本章では TNR+AF のアルゴリズムの詳細を調査し，pgrouting に実装を行う．

3.1 TNR+AF の詳細

次のように記号を定義する．

$A^{\rightarrow}(s)$: T の内，始点 s から最初に訪れる
可能性のあるノードの集合

$A^{\leftarrow}(t)$: T の内，終点 t に最後に訪れる
可能性のあるノードの集合

$f^{\rightarrow}_{s,u}(x)$: 領域 x に存在するノード $v \in T$ について，
$d(s,u)+d(u,v)$ が $min\{d(s,u')+d(u',v)|u' \in A^{\rightarrow}(s)\}$
と等しい場合のみ $true$ となる $flag\ vector$

TNR+AF の詳細は以下の通りである．

- 前処理
  1. Transit Node を選択する．
  2. 領域を p 個に分割する．
  3. 境界ノードを選択する．
  4. 各ノード $s \in V$ と境界ノード $b \in B$ について，$d(s,u)+d(u,b)$ を最小にする $access\ nodes$ $u \in A^{\rightarrow}(s)$ を決定し，$f^{\rightarrow}_{s,u}(r(b))$ を $true$ にする．
  5. 各ノード $s \in V$ と各 $access\ nodes$ $u \in A^{\rightarrow}(s)$ について，$f^{\rightarrow}_{s,u}(r(u))$ を $true$ にする．
  6. ステップ 4 と 5 を逆方向でも行う．
- 探索
  1. t の全ての $backward\ access\ nodes$ を考え，それぞれの $u \in A^{\leftarrow}(t)$ について $f_t(r(u)) = true$ となるような $flag\ vector$ f_t を形成する．
  2. $forward\ access\ nodes$ u と s について，$f^{\rightarrow}(s,u)$ と f_t のビット演算が 0 でないものだけを考え，このような集合を $A^{\rightarrow\prime}(s)$ とする．それぞれの $u \in A^{\rightarrow\prime}(s)$ について，$f_s(r(u)) = true$ となるような $flag\ vector$ f_s を形成する．
  3. f_s を用いてステップ 2 と同様に部分集合 $A^{\leftarrow\prime}(t) \subseteq A^{\leftarrow}(t)$ を決定する．
  4. $|A^{\rightarrow\prime}(s)| \times |A^{\leftarrow\prime}(t)|$ の探索を実行する．

以下に TNR+AF の概念図を示す.

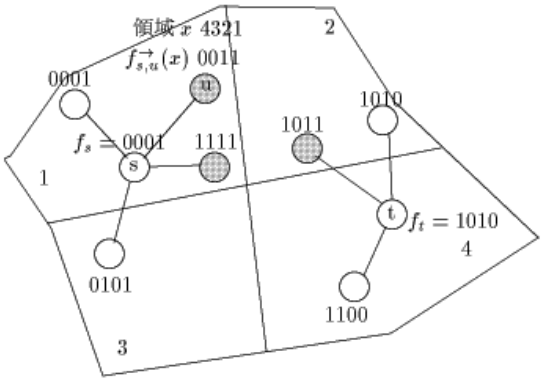

図 1 TNR+AF の概要

3.2 pgrouting への実装

次に，提案した手法を pgrouting に実装する．pgrouting はオープンソースの経路検索エンジンであり，PostgreSQL/PostGIS 上で動作する (http://postgis.refractions.net/).

pgrouting では探索のためのプログラムは C で記述されており，sql ファイルで関数の定義などを行っている．よって前章で提案した手法も C で実装する．

前処理の実装

- 領域の分割
  領域の分割方法として参考文献 [4] では，グリッド状の分割や Kd-Tree などが挙げられている．実験の結果それぞれの領域にほぼ同数ずつのノードが収まるように分割することができる Kd-Tree が有効であることがわかっているが，最もシンプルなグリッド状の分割と比較して探索速度は 1.1 倍から 1.2 倍程度しか変化しない．そこで本研究ではプログラムの容易さも考慮して領域の分割はグリッド状に行うものとする．前処理において変数 p を定め，$p \times p$ 個の領域に分割する．
- Transit Node の選択
  参考文献 [3] の sweep line アルゴリズムを用いる．このアルゴリズムは以下のように計算される．グリッド状にグラフを分割した領域のうち，縦のラインの 1 つを sweep line とする． sweep line と交差するエッジの 1 端を v とし，ノード v と次の図のような位置関係にあるセルに着目する．
  着目したセルの内，v の左側を C_{left}，右側を C_{right} とする．C_{left} の境界ノードと v，v と C_{right} の最短距離の組み合わせを計算し，合計が最小となるときの v を Transit Node として記憶する． sweep line をずらしながらこれを繰り返す．横の sweep line についても同様のことを行う．

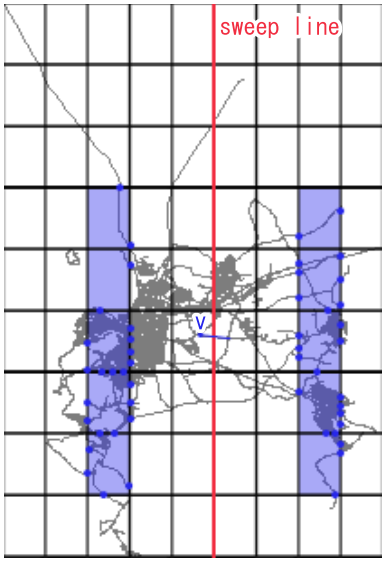

図 2 sweep line の説明 (縦)

- Distance Table の作成
  選択した Transit Node 間の最短経路と距離を格納するために Distance Table を作成する．Distance Table の項目は tn_id1，tn_id2，$distance$，the_geom_tt の 4 つである．tn_id1 と tn_id2 は Transit Node の ID である．$distance$ は tn_id1 と tn_id2 間の距離であり， the_geom_tt は tn_id1 と tn_id2 間の経路の情報が記述されたジオメトリである．
- $access\ nodes$ 及び $flag\ vector$ の決定
  前述の手順で $access\ nodes$ 及び $flag\ vector$ を決定し，新しいテーブルを作成し，格納する．テーブルの項目は s_id，u_id，$flag\ vector$ の 3 つである．s_id はグラフ上のノード s の ID，u_id は s のアクセスノードの ID である．$flag\ vector$ は領域数分のビット数を確保し，デフォルトでは全て $0(false)$ とする．$f_{s,u}^{\rightarrow}(i)$ が $true$ となるとき，前から i 番目のビットを $1(true)$ に設定する．

探索の実装

探索を行うために，探索結果格納用テーブルを作成する．テーブルの項目は s_id，t_id，the_geom の 3 つである．探索プログラムを実行すると，テーブルに始点 s_id から終点 t_id までの探索結果のジオメトリを格納する．これを参照することで探索結果を取得することができる．

4 TNR+AF の性能評価

本研究で実装した探索手法の有効性を示すために実験を行う．実験では実装した TNR+AF と，従来の pgrouting によるダイクストラ法と A*探索とを同一環境下で実行する．ただし，TNR + AF については領域の分割数を 12×12，16×16，20×20 の 3 つを考える．

実験に用いる地図データは南アフリカ，ケープタウンの地図データ (http://www.pgrouting.org/docs/foss4g2008/index.html)

である．エッジ数は 40585，ノード数は 30523 であり，これをマップ 1 とする．マップ 1 をエッジ数 20292，ノード数 25054 に省いたものをマップ 2 とし，この 2 つを実験に用いる．はじめに TNR+AF(12×12)，TNR+AF(16×16)，TNR+AF(20×20) の 3 つについて前処理の時間を計測する．結果は以下の通りである．

表 2　前処理時間の計測結果

	前処理時間 [min]	
	マップ 1	マップ 2
TNR+AF(12×12)	197	71
TNR+AF(16×16)	475	191
TNR+AF(20×20)	913	422

次にそれぞれのマップについて，マップ中から始点と終点のノードの ID をランダムに決定し，その間の最短経路探索を各手法で実行し，それを 1 万回繰り返した．探索回数と平均探索時間の関係は以下のグラフの通りである．

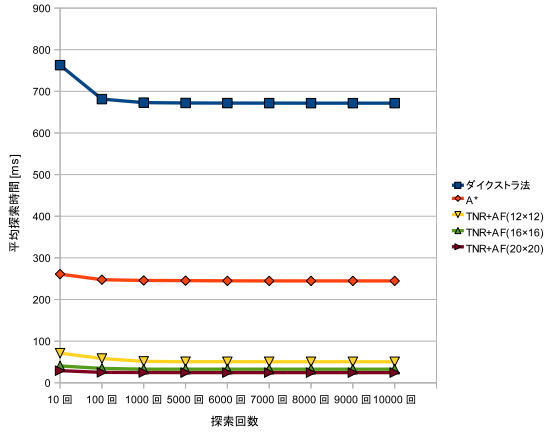

図 3　平均探索時間の計測結果 (マップ 1)

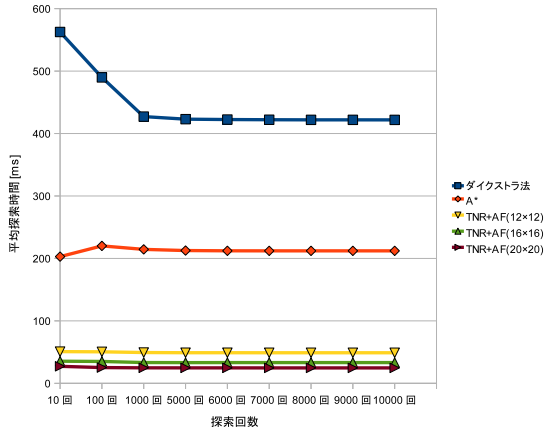

図 4　平均探索時間の計測結果 (マップ 2)

この結果からどの手法についても 1 万回の時点で平均探索時間は充分収束していることがわかる．よって以降では 1 万回の時点での平均探索時間をそれぞれの手法の探索時間とする．

それぞれの手法の 1 万回の時点での平均探索時間は以下のとおりである．

表 3　探索時間の計測結果

	探索時間 [ms]	
	マップ 1	マップ 2
ダイクストラ法	671.61	421.90
A*	244.65	212.14
TNR+AF(12×12)	50.44	48.94
TNR+AF(16×16)	32.59	33.17
TNR+AF(20×20)	24.33	24.68

各手法について探索を繰り返した場合，前処理時間を P，探索時間を Q，探索回数を N，総探索時間を T とすると，

$$T = P + Q \times N \tag{1}$$

となる．以下に各手法について，T と N の関係を示す．

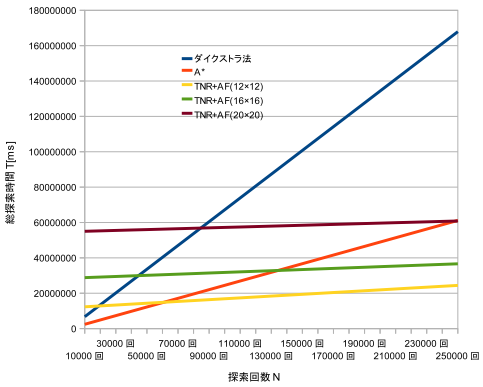

図 5　総探索時間の推移 (マップ 1)

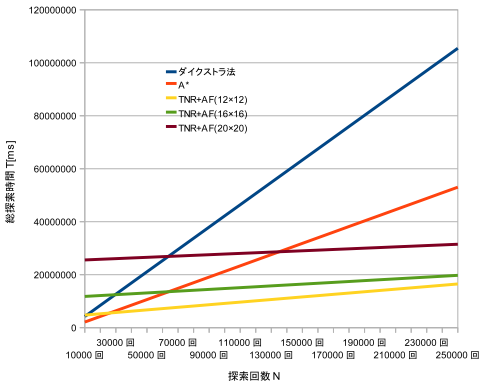

図 6　総探索時間の推移 (マップ 2)

以上から，どちらのマップにおいても総探索時間で既存の手法を下回るのは TNR+AF(12 × 12) が最も早く，TNR+AF(20 × 20) が最も遅かった．また最も既存の手法を下回るのが遅い TNR+AF(20 × 20) でも，マップ 1 では 248639 回，マップ 2 では 135069 回以上の探索で A* の総探索時間を下回ることがわかった．

最後に本論で議論した TNR+AF を用いた探索手法を Mapserver を用いて可視化した．Mapserver とは，WEB マッピングアプリケーションのためのオープンソースの GIS プラットフォームである (http://mapserver.org/)．Mapserver は PostGIS と接続可能で，クエリを実行しその結果を WEB ページ上に表示することが可能である．

本研究では下図のようにトップページで始点，終点の ID を取得し，経路探索用ページで地図データとクエリの結果を表示する．

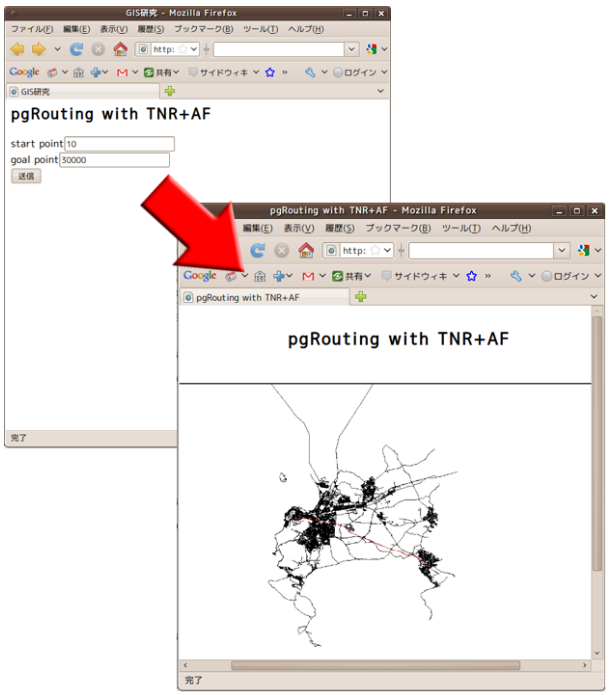

図 7 Mapserver による探索結果の可視化

5 考察

以上の実験結果から，ケープタウンのマップデータにおいては，探索回数が 248639 回を超えると 12 × 12 の TNR+AF が有効であることがわかった．領域の分割数が多い場合前処理には更に時間が掛かるが，探索時間そのものは削減できているため，探索回数が更に多い場合には領域の分割数を多くするべきだと考えられる．計算の結果探索回数が 934454 回以上なら 16 × 16，3181599 回以上なら 20 × 20 に領域を分割するのが有効であるとわかった．

これらの値はマップのサイズによって変化する．エッジ数とノード数を削減したマップ 2 においては，マップ 1 に比べて前処理に要する時間が減少し，探索時間には大きな変化がなかったため，繰り返す探索回数が少なくても TNR+AF は有効になることがわかった．

また，異なる地図において TNR+AF を用いるとき，前処理において分割した領域のそれぞれにどの程度のノードが含まれているかが目安になると考えられる．ケープタウンの場合においては，分割数が 12 × 12 のとき平均 212.0，16×16 のとき平均 119.2，20×20 のとき平均 76.3 のノードが 1 つの領域に含まれている．これらの値を目安に領域の分割が粗すぎず，細かすぎないように考慮することで有効な探索が可能であると考えられる．

6 結論

本論文の結果 TNR+AF は従来の手法に比べて探索時間を削減できていることや，探索を繰り返すことで前処理に要した時間を埋められることがわかった．前処理を用いた最短経路探索手法は前処理時間と探索時間がトレードオフの関係になっており，一概にどの手法が優れているということは出来ない．しかし探索時間の削減という点に着目する場合，本論文で採用した TNR+AF は非常に優れていることがわかる．そのため WEB ルーティングサービスのような不特定多数の人間によって何度も探索が行われる場合には，今回の手法の利点を生かすことができると考えられる．今後の課題として，先行研究と比較して特に前処理に時間が掛かってしまっているため，これを改善するように適切なアルゴリズムを考えるべきであると思われる．

参考文献

[1] A. Goldberg and C. Harrelson, “Computing the Shortest Path: A* Search Meets Graph Theory,” Technical Report MSR-TR-2004-24, Microsoft Research, 2004.

[2] D. Schultes, “Route Planning in Road Networks,” Ph.D thesis, Universitat Karlsruhe, 2008.

[3] H. Bast, S. Funke, and D. Matijevic, “TRANSIT: ultrafast shortest-path queries with linear-time preprocessing,” Proc. of 9th DIMACS Implementation Challenge, 2006. (http://www.dis.uniroma1.it/ challenge9/)

[4] M. Hilger, E. Kohler, R. Mohring, and H. Schilling, “Fast Point-to-Point Shortest Path Computations with Arc-Flags,” Proc. of 9th DIMACS Implementation Challenge, 2006. (http://www.dis.uniroma1.it/ challenge9/)

[5] P. Sanders and D. Schultes, “Engineering highway hierarchies,” Proc. of 14th European Symposium on Algorithms, LNCS, vol.4168, pp.804-816, Springer, 2006.

[6] R. Bauer et al, “Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra’s Algorithm,” Proc. of 7th Workshop on Experimental Algorithms (WEA), LNCS, vol.5038, pp.303-318, Springer, Heidelberg 2008.