二次記憶上の道路地図データベースに対する最短路探索手法

2R－8

鈴木 尚志[†]　大保 信夫[††]

[†]筑波大学 理工学研究科　[††]筑波大学 電子・情報工学系

1 はじめに

今日の情報の多様化に伴い，より複雑な情報を効率良く管理する高度なデータベースシステムを開発しようとする動きが起こってきた．その様なデータベースシステムの一つとして，**道路地図データベースシステム**がある．

道路地図データベースシステムでユーザの利用頻度が高い検索の一つに，地図上の２点を結ぶ最短路を求める**最短路探索**がある．最短路探索に関しては既に様々な手法が提案されているが，それらは道路データが全て主記憶上に置かれていることを想定しており，二次記憶への効率良いアクセスを想定した探索アルゴリズムは知られていない．本論文では，二次記憶に格納した道路データに対して効率良く最短路探索を行なう手法を提案し，その効率について検討する．

2 最短路探索

道路地図は二次元平面上に図形として表現されるが，描かれている道路網は，交差点をノード，交差点間を結ぶ道路をエッジ，道路の長さや移動時間などをエッジの重みとするグラフと見ることが出来る．この時，任意の交差点 u,v 間の最短路探索とは，u から v へ至る経路の中で，経路長 (重みの和) が最小のものを見つける探索のことを言う．

最短路を求める手法は種々あるが，その中でも代表的なものにダイクストラのアルゴリズムがある．

2.1 ダイクストラのアルゴリズム

始点 (現在地) と終点 (目的地) を指定すると，両者を結ぶ経路上の各ノードについて，始点からの経路長が小さい順に訪問して探索する．つまり，始点との空間的距離が小さいノード，いわゆる“始点に近いノード”から順に訪問し，終点を訪問したところで探索が終了する．

ところが，次に訪問するノードを選択する際，そのノードデータが主記憶上にあるか判定しないので，探索順序は二次記憶上の格納構造とは無関係に決められる．このため，格納構造によっては，探索を進めるために必要なノードデータが主記憶上に存在せず，二次記憶から読み込む回数が増える．二次記憶のデータの読み込みは主記憶内データのやりとりに比べてコストが高いため，二次記憶への頻繁なアクセスは探索時間を増大させる．

また，このアルゴリズムの探索領域 (訪問されたノードの存在する空間領域) は，始点と終点を半径とする円の内側である．よって，始点と終点の距離が大きくなると，最短路上にないノードへの訪問数も増加してしまう．

3 Domain Encoding

二次記憶上の道路データに対して効率良く最短路を求めるために，次の二つの点を考慮した探索アルゴリズムが有効と考えられる．

1. 二次記憶へのアクセス回数を少なくする．
2. 探索領域を縮小する．

本研究では，巨大なグラフを扱う手法として提案されている **Domain Encoding** [?] に基づく方法を提案する．

Domain Encoding は，巨大な道路網のグラフを幾つかの部分グラフ (**ドメイン**) に分割し，各ドメインの情報を二次記憶上の個別のファイルに格納する．ドメインは，**中心** (center) と呼ばれるノードで識別される．ドメイン内の各ノードは互いに連結しているが，他のドメインに属するノードとも連結するノード (**境界**ノード) も存在する．また，中心から最も遠い境界ノードと中心との間の最短路長を**半径** (radius) と呼ぶ．

3.1 DS アルゴリズム

Domain Encoding を用いて道路データを二次記憶に格納した時，最短路探索での二次記憶から主記憶へのデータ転送はドメイン単位で行なわれる．そこで，同一ドメイン内のノードを連続的に訪問し，ページフォルトを少なくすることを目的にダイクストラのアルゴリズムを変更した探索手法を提案する．以下，この手法を **DS (Domain Search)** アルゴリズムと呼ぶことにする．

また，DS アルゴリズムでは探索領域を縮小する目的で，始点からの経路長がある上限値 (upper bound) を越えないノードのみを訪問するよう，条件が加えられている．その上限値を計算するため，domain encoding で用いられる次のパラメータ予め計算しておく．

・全てのドメインの中心間を結ぶ最短路長
・全てのドメインの半径

そして，ノードを訪問するための新たな条件を次に示す．

グラフ中に二つのドメイン D_1, D_2 があり，各々の中心を c_1, c_2 とおく．D_1 内のノード p_1 と D_2 内のノード p_2 の間の最短路長を求める時，他のドメイン D_3 に属するノード p_3 を訪問するためには，始点からの最短路長 p_1p_3 が次の不等式を満たさなければならない．

$$p_1p_3 < (c_1c_2 - c_3c_2) + p_1c_1 + p_2c_2 + c_2p_2 + r_3 \quad (1)$$

なお，$c_1c_2, c_3c_2, p_1c_1, p_2c_2, c_2p_2$ はそれぞれ２ノード間

The Shortest Path Searching Method for Road Map Database
Takashi SUZUKI[†], Nobuo OHBO[††]
[†]Master's Degree Program in Science and Engineering, Univ. of Tsukuba
[††]Institute of Information Sciences and Electronics, Univ of Tsukuba

の最短路長を，r_3 は p_3 が属するドメインの半径を表す．このうち，c_1c_2, c_3c_2 と r_3 は予め計算されており，p_1c_1, p_2c_2, c_2p_2 は探索時に計算する．

二次記憶とのデータのやりとりはドメイン単位で行なわれるため，探索領域の縮小によって二次記憶へのアクセス回数も減少する．

ここで，DS アルゴリズムの探索手順を表 1 に示す．道路網のグラフはドメインの集合 D から成り，D の各要素は，ノード集合 N，エッジ集合 E，コスト (始点からの最短路長) の集合 $Cost$ の情報を持つものとする．最短路探索の始点は s，終点を g と表す．

```
Procedure DS (D, s, g)
begin
  foreach d in D do
     foreach u in N do  { Cost(s,u) = ∞, path(s,u)  = null, }
  read domain_file that includes s,
  frontierSet  = {s},   anotherpageSet  = φ,
  successor  = s, Cost(s,successor) = 0,
  while (successor ≠ d) and (Cost(s,successor) < Cost(s,g))
     { if not_empty(frontierSet) then
         select u from frontierSet with minimun Cost(s,u);
       else if not_empty(anotherpageSet) then
         { select u from anotherpageSet with minimun Cost(s,u),
           replace domain_file that includes u, }
       else terminate,
       foreach <v,Cost(u,v)> in u.adjacencyList
          { if Cost(s,v) > Cost(s,u) + Cost(u,v) then
          { Cost(s,v) = Cost(s,u) + Cost(u,v), path(s,v)  = path(s,u) + E(u,v), }
           if in(v,D)
              { if not_in(v, frontierSet) then
                  frontierSet  = frontierSet + {v}, }
            else
              { if not_in(v,anotherpageSet) and Cost(s,v) < upperbound then
                  anotherpageSet  = anotherpageSet + {v}, }
       if u == successor then
          select successor from (frontierSet ∪ anotherpageSet)
                        with minimum Cost(s,successor),
     }
end,
```

表 1:DS アルゴリズムの探索手順

表中の frontierSet, anotherpageSet は，共に「訪問済みノードに連結した未訪問ノードの集合」を意味し，次に訪問するノードの候補となる．前者の各要素は探索中のノードと同じドメインに属しているが，後者の各要素は探索中のノードとは異なるドメインに属し，且つ式 1 の不等式を満たすもののみである．また，ダイクストラのアルゴリズムとは異なり，DS アルゴリズムの探索順序では，厳密解である経路上のノードより先に終点を訪問し，探索を終了してしまう恐れがある．よって，厳密解上のノードを全て訪問するまで探索が継続するよう，successor($\subset N \subset D$) を用いた終了条件を与えた．

4 DS アルゴリズムの効率評価

DS アルゴリズムとダイクストラのアルゴリズムとの探索効率を比較するため，実際の道路地図データ，及びノード・エッジが格子状に分布するモデルを用いて，任意の 2 点間の最短路を計算する実験を行なった．ただし，格子モデルでは実道路データと同様の結果が得られたので，ここでは実道路データでの実験結果を示す．

4.1 データ及び実験条件

使用したデータは，日本デジタル道路地図データベース標準 (DRM)[?] からつくば市，土浦市近辺の道路データ (ノード数 6445，エッジ数 9666) である．各ノードの空間的位置関係により二次元の座標値を与え，座標値によって 25 個のドメインに分割し，各ドメインを二次記憶の個別のファイルに格納した．各エッジの重みは実測値に基づいた数値を与えた．また，主記憶には 1 ドメインのデータ長を 1 フレームとする 10 フレーム分のバッファを確保し，その管理は LRU アルゴリズムを用いて行なった．そして，これらのデータに対し

1. 始点・終点をランダムに与える．
2. 二つのアルゴリズムで最短路を求める．

という行程を 500 回繰り返した．

4.2 結果と考察

まず，探索時のファイルアクセス回数を調べた結果を図?? に示す．横軸は求められた最短経路長で，縦軸がファイルアクセス回数を表す．

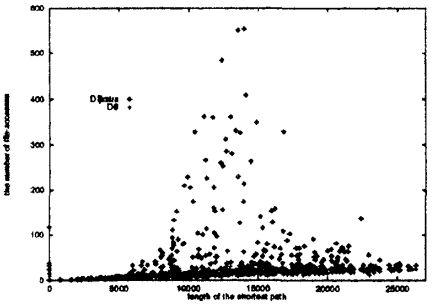

図 1: 探索時のファイルアクセス回数

探索領域が 10 ドメイン (主記憶のバッファサイズ) を越えると，ダイクストラのアルゴリズムを用いた場合のファイルアクセス回数は，探索終了までに訪問したノード数に比例している．これは，訪問中のノードと次に訪問するノードが異なるドメインに属していても，それを考慮せずに探索を進め，ページフォルトを引き起こすのが原因である．それに対して，DS アルゴリズムを用いた場合では，ファイルアクセス数は探索領域の大きさに比例している．図?? に示す結果において，二つのアルゴリズムで最も格差が大きい所では，DS アルゴリズムを用いた場合のファイルアクセス数がダイクストラのアルゴリズムの場合の約 30 分の 1 であった．

尚，最短路長 (横軸) が最大に近付くと，ダイクストラのアルゴリズムを用いた時のファイルアクセス数が減少している．これは，最短路長で約 27000 の長さを対角線とする正方形領域全体にノードが分布しているため，結果的に無駄なファイルアクセスが抑えられたためである．

参考文献

[1] Rakesh Agrawal and H. V. Jagadish. *Algorithms for Searching Massive Graphs.* IEEE Transactions on Knowledge and Data Engineering, vol.6, No.2, pp.225-237, 1994.

[2] 日本デジタル道路地図協会．「全国デジタル道路地図データベース標準 第 2.1 版」．日本デジタル道路地図協会, 1991