DEIM Forum 2013 E8-4

空間分割を用いた経路スカイライン探査法の提案

玉置　和也†　　上島　紳一†

† 関西大学総合情報学研究科　〒 569–1095 大阪府高槻市霊仙寺町 2-1-1
E-mail: †k711829@edu.kansai-u.ac.jp, ††ueshima@kansai-u.ac.jp

あらまし　本稿では，多次元コストグラフに沿って移動した際の経路コストに対する経路スカイラインの抽出手法について議論する．経路スカイラインは，多次元データに対するスカイラインと異なり，グラフ上の探査処理を必要とする．ここでは，経路スカイラインの任意の部分経路がパレート最適であることに着目した 2 段階から成るアルゴリズムを提案する．まず，前処理によりグラフ分割により得られた部分領域に対して，探査処理によって経路スカイラインの部分経路候補集合を求め，次に，探査実行処理として経路スカイラインを構成的に計算する．両段階の探査では経路間の支配関係に基づき，探査空間の刈込み処理を行って効率的に抽出する．本手法の有効性を確認するために，数値地図を用いてシミュレーションを行い，前処理時間，探査実行時間，属性数の影響などについて検討し，関連手法である Kriegel らの手法 [7] と比較する．

キーワード　経路スカイライン，スカイライン問合せ，空間データベース，空間分割，経路探査

Route Skyline Query Processing using Spatial Tessellation

Kazuya TAMAKI† and Shinichi UESHIMA†

† Graduate School of Informatics, Kansai University
2-1-1 Ryozenji, Takatsuki, Osaka, 569–1095 Japan
E-mail: †k711829@edu.kansai-u.ac.jp, ††ueshima@kansai-u.ac.jp

Abstract　In this paper, route skyline query processing problem is studied over multidimensional cost graphs. Route skyline problem is to extract the totality of routes between specified start/goal nodes, that minimize given linear cost function with non-negative coefficients. Route skyline problem requires route search on the given graph in order to derive skyline candidates, while data set are explicitly given beforehand in skyline problems for multi-dimensional data points. Here the authors propose a two-stage algorithm of preprocessing and search processing to extract route skyline candidates, utilizing the fact that arbitrary partial routes of skyline are also skyline. We use spatial tessellation of the target graph, and generate route skyline in a bottom-up fashion, by combining route skyline candidates in partitioned regions. Numerical simulations for digital road maps are also provided to verify the efficiency of the proposed algorithm quantitatively and to compare it with ARSC in Kriegel et al [7].

Key words　Route Skyline, Skyline Query, Spatial Database, Spatial Tessellation, Route Search

1. はじめに

近年，多次元データ集合に対して，利用者の選好を考慮した最適な部分集合を求めるスカイライン問合せに関する研究が盛んである．このスカイライン問合せでは，対象となる多次元データ集合が明示的に与えられ，データ間の支配関係を調査しながら，他のデータに支配されない集合を解として抽出する．また高次元データ集合に対しては，効率的な計算方法や並列演算処理環境などの手法を用いて，問合せ処理を行う方法が提案されている [1,5,6].

これに対して [7] では，正の重みをエッジコストとして持つ多次元グラフに対して，経路間の支配関係に基づく経路スカイライン問題が提案されている．経路スカイライン問合せでは，多次元データの場合と異なり，予め 2 地点間を結ぶ経路集合全体が明示的には与えられておらず，グラフ上で探査を行う必要がある．しかし，属性数が多い場合や大規模グラフの場合には，探査時間やファイルアクセスなどの計算コストが増大する [7].

本稿ではこの点を解決するため，経路スカイライン内の経路の任意の部分経路がパレート最適になっている点に着目した抽出アルゴリズムを提案する．提案手法は生成処理と探査実行処理の 2 段階から構成される．生成処理ではグラフを空間分割し，その部分領域内で経路スカイラインの候補集合を予め求めてお

く．探査実行処理では与えられた出発地点と目的地点に対して候補集合上で最適な経路を組合せる処理を実行して解集合を構成的に抽出している．

2 章で関連研究について述べ，3 章で経路スカイライン問題の定式化を行い，4 章で提案アルゴリズムについて述べ，5 章で数値シミュレーションによる提案手法の評価，6 章でまとめる．

2. 関 連 研 究

多次元データ集合に対するスカイライン問合せは，利用者選好を係数とする評価関数に対して最適解を短時間で提供できる利点を持つため，様々な効率化手法が提案されている [1–5]．

Börzsönyil ら [1] は，スカイラインについて最初に議論した論文で，データベース処理の観点から，主記憶への読込バッファと外部記憶を利用したアルゴリズムとして，データ集合の分割を用いたスカイライン点の抽出法を提案している．

また，スカイライン集合は，データ間の支配関係によって定まるため，支配関係の調査の効率化を図る様々なアルゴリズムが提案されている．[2,3] では，対象とする多次元データ集合を水平型や垂直型に分割し，部分集合毎にスカイライン点集合を求めて統合する方法が提案され，分散型，並列型システム等で実行する方法が議論されている．[4,5] では，対象データを超平面に射影して角度をもとに分割する手法を提案している．

一方，Sharifzadeh ら [6] は，ユークリッド空間内のオブジェクト集合を対象とした空間スカイライン問題を議論している．ここでは，質問点集合からオブジェクト集合までの距離を用いてオブジェクト間の支配関係を定義し，空間のボロノイ分割を用いた抽出手法を提案している．

さらに，Kriegel ら [7] は，複数属性付きエッジからなる多次元コストグラフ上で，利用者選好を係数とする経路コスト関数に対して経路スカイラインを議論している．ここでは，空間埋込み法を用いて，グラフ上に複数の参照点を配置し，参照点への距離見積りを指標として作成する事前処理を行って，探査実行時における経路の刈込みを可能にする手法を提案している．[8,9] では，利用者への利便性を高めた GUI を構築し実応用へ適応し易く工夫している．しかし，同時に参照点数や対象データの次元数等に比例して指標の計算量の増加が指摘されている [7]．

本稿ではこれに対し，予めグラフを分割して，部分領域毎に経路探査を実行し解候補の部分となる経路集合を求めておくことで，前処理と実行時の処理の負荷を分散させ，実行時の探査実行処理を軽量化する方法を提案している．これにより，探査実行時に，前処理で得られた解候補集合内の経路を組合せる処理を効率的に実行し，解経路集合を構成的に抽出できる．

3. 経路スカイライン問題

ここでは，経路スカイライン問題の定式化を行い，経路スカイラインの特徴について述べる．

3.1 問 題 設 定

[定義 1]（多次元コストグラフ） 多次元コストグラフにおいて，V をノード集合，E をエッジ集合（$E \subset V \times V$），$W \subset \mathbb{R}_{+}^{d}$ を

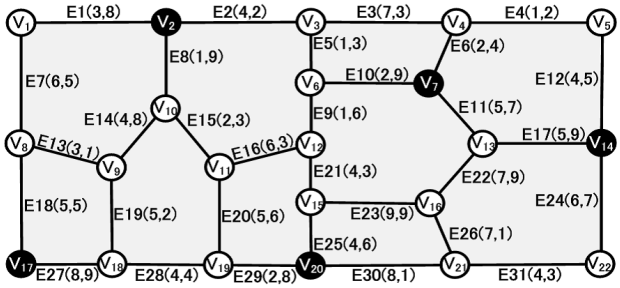

図 1　2 次元コストグラフ $G(V,E,W)$ の例 $(d=2)$

$e \in E$ の非負の値を持つ d 次元のコスト値 (属性値) とするグラフを多次元コストグラフ $G(V,E,W)$ という (以下，単に G とも書く)．本稿で G は無向グラフとする．

G を道路網とみなす場合は，交差点をノード，交差点を結ぶ道路片をエッジ，道路片のコストをエッジの属性値と考える．属性数 2 に対する G の例を図 1 に示す．G 上で，利用者は出発地点 v_s から目的地点 v_t に向かって移動する時，経路コストを次のように定義する．

[定義 2]（経路と経路コスト） $G(V,E,W)$ 上の経路 p を連接するノードの列 $p=(v_1,v_2,\ldots,v_k), v_i \in V$ とし，構成エッジ (v_i,v_{i+1}) の第 l 番目のコスト $w((v_i,v_{i+1}))_l (1 \leqq i < k, l = 1,2,\ldots,d)$ を用いて p の第 l 番目のコスト $cost^l(p)$ を式 (1) のように定義する．

$$cost^l(p) = \sum_{i=0}^{k-1} w((v_i,v_{i+1}))_l \tag{1}$$

つまり経路 p は，経路の構成ノードを繋ぐ隣接エッジのコストの和である．この時，p の d 次元コストベクトルは，

$$cost(p) \;=\; col(cost^1(p), cost^2(p), \ldots, cost^l(p)) \tag{2}$$

となる．本稿では，G 上で，より小さいコストを持つ経路が他より優れている経路とする．

[定義 3]（利用者選好と選好関数） 経路に対する d 次元の利用者の選好ベクトル $\Pi \subseteqq \mathbb{R}_{\geqq 0}^{d}$ と，経路 p に関する選好関数 $Pref_{\Pi}(p)$ をそれぞれ式 (3), (4) のように定義する．

$$\Pi = col(\pi_1,\pi_2,\ldots,\pi_d) \in \mathbb{R}_{\geqq 0}^{d} \backslash \{\vec{0}\} \tag{3}$$

$$Pref_{\Pi}(p) = \sum_{l=1}^{d} \pi_l \cdot cost^l(p) =< \Pi, cost(p) > \tag{4}$$

[定義 4]（経路間の支配関係） $G(V,E,W)$ 上で，出発地点ノード v_s と目的地点ノード v_t の 2 地点間を繋ぐすべての経路を $P(v_s,v_t) = \{(v_s,\ldots,v_t)\})$ とする．$p,q \in P(v_s,v_t)$ に対して，

$$\begin{aligned} &cost^l(p) < cost^l(q) \quad (\exists l, 1 \leqq l \leqq d) \\ &cost^j(p) \leqq cost^j(q) \quad (\forall 1 \leqq j \leqq d \wedge j \neq l) \end{aligned} \tag{5}$$

が成り立つ時，経路 p が経路 q を支配しているという．式 (5) で経路 p は，第 l 属性について優れており，他の属性 j に関しても同等か優れていることを意味する．

Note: 式 (4) の選好関数の定義より，Π は各経路の経路コス

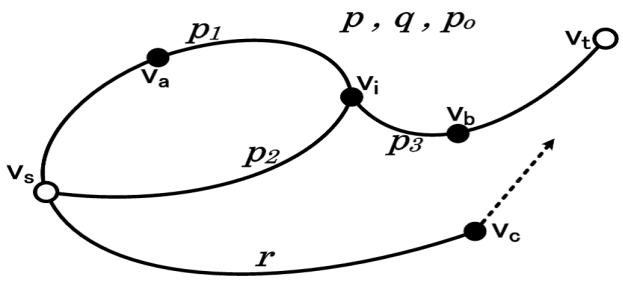

図 2　部分経路スカイライン

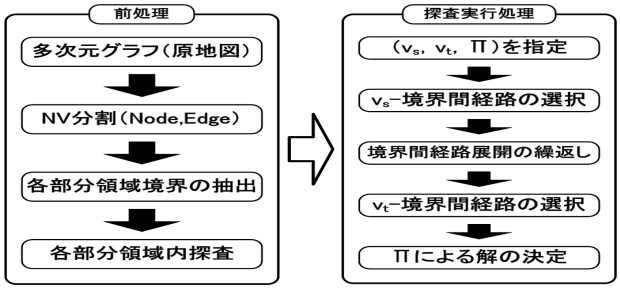

図 3　経路スカイライン抽出の流れ

トに対して次元毎に均一に影響を及ぼすため，Π の値を変更しても経路間の支配関係に影響しないことがわかる．この経路間の支配関係を用い，次のように経路スカイラインを定義する．

[定義 5] (経路スカイライン) $G(V,E,W)$ 上に出発地点 v_s と目的地点 v_t が与えられた時，経路集合 $P(v_s,v_t)$ 内で他の経路に支配されない経路集合を**経路スカイライン** $RS(v_s,v_t)$ という．

また，経路集合 $P(v_s,v_t)$ から，他の経路に支配されない経路スカイラインを抽出する問題を経路スカイライン探査問題という．本稿では，Π に対して選好関数を最小化する事を考え，経路スカイラインを求める．この時，次の補題が成立する．

補題：$\forall\breve{p} \in RS(v_s,v_t)$ に対して，

$$\exists \Pi \in \mathbb{R}^d \quad s.t. \quad Pref_{\Pi}(\breve{p}) \leqq Pref_{\Pi}(p)(\forall p \in P(v_s,v_t))$$

つまり経路スカイライン集合 $RS(v_s,v_t)$ に含まれる任意の経路 $\breve{p}$ をとると，$P(v_s,v_t)$ のすべての経路と比較して，より小さいか，または等しい選好関数の値を持つ選好ベクトル Π が存在する．経路スカイライン探査問題は，利用者が与える選好に対するパレート最適経路集合を見つけ出すことである．

3.2　経路スカイラインの特徴

経路スカイラインに含まれる経路は，次の 3 つの性質を持つ．

(性質 1)　$G(V,E,W)$ 上の経路スカイライン $RS(v_s,v_t)$ に属する経路の任意の部分経路は，部分経路スカイラインである．例えば，図 2 で，$p=(v_s,v_a,v_i,v_b,v_t) \in RS(v_s,v_t)$ の時，p の部分経路 $p_1=(v_s,v_a,v_i) \in RS(v_s,v_i)$ であり，他の $p_2=(v_s,v_i) \in P(v_s,v_i)$ に支配されない．同様に，$p_3=(v_a,v_i,v_b) \in RS(v_a,v_b)$ などが成り立つ．

(性質 2)　$G(V,E,W)$ 上の 2 点 v_s,v_t に対して，$RS(v_s,v_t)=RS(v_t,v_s)$ である．例えば，図 2 で，$p=(v_s,v_a,v_i,v_b,v_t) \in RS(v_s,v_t)$ に対して逆向き経路 $q=(v_t,v_b,v_i,v_a,v_s) \in RS(v_t,v_s)$ である．

(性質 3)　v_s から出発した任意の長さを持つ経路 r が，ある経路スカイライン $p_o \in RS(v_s,v_t)$ に支配される時，r を v_t まで延長しても p_o に支配される．例えば，図 2 で，$r=(v_s,v_c)$ が p_o に支配されれば，r を延長して v_t に到達しても p_o に支配される．

[注]　異なる 2 つの経路スカイラインを結合した経路はスカイラインになるとは限らない．

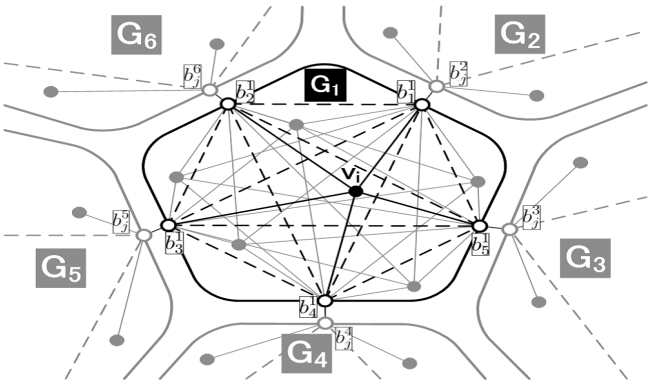

図 4　G の分割と G_1 内の各経路スカイラインの例

4.　経路スカイライン抽出手順

4.1　探査の 2 段階構成

経路スカイラインを求めるためには G 上の 2 点間に対し経路探査が必要であり，G の規模に比例し計算量は増大する．このため G が大規模の場合，G を分割し各部分領域内で局所的に 2 点間の経路探査を行うことで，組合せの数を削減する手法が有効であると考えられる．そのため本稿では，3.2 節 (性質 1) で述べた，経路スカイラインの任意の部分経路が部分経路スカイラインである点に着目し，経路探査の計算量を縮減する方法について考える．提案手法による経路スカイライン抽出の流れを図 3 に示す．提案手法は (1) 前処理と，(2) 探査実行処理の 2 段階で構成する．前処理として各部分領域内で探査処理を行い解候補集合を抽出しておき，探査実行処理として各部分領域を跨ぎ解候補集合を組合せながら延長し経路を生成する処理を行う．また，探査途中の経路間の支配関係のモニタリングでは，3.2 節の各性質に対応した探査停止基準を設け，探査処理 (連結処理) を停止する．これより G 上の探査空間を刈込むことができる．

4.2　前　処　理

前処理は次の 3 段階 [1]，[2]，[3] からなる．

[1] $G(V,E,W)$ の NV 分割

$G(V,E,W)$ に対して NV 分割法 [10] を用い，部分グラフ集合に分割する．まず，$G(V,E,W)$ 上のノード集合 V から確率的に母点を選択する．母点から他のノードに対して並列ダイクストラ法を実行し，G を部分グラフ群 $\{G_1,G_2,\ldots,G_n\}$(n は分割数) に分割する (図 4 の G_1〜G_6)．ここでは，G(V,E,W) のエッジが複数の属性を持つため，選択した母点から各ノード

Class Node	Class Boundary	Class Route
Int nodeID;	Int boundaryID;	Int routeID;
Int motherID;	Int nodeRight;	Int srcNode;
Int hop;	Int nodeLeft;	Int dstNode;
Double X,Y;	Int rightMotherID;	Int flag;
arrayList(Route) partRoute;	Int leftMotherID;	Double cost1,...,5
arrayList(Route) routeTemp;	Double cost1,...,5	arrayList(Int) nodeList;

表 1　ノード，境界，経路のデータ構造

に至る距離の指標としてホップ数 (エッジ数) を用いる．これにより，各 G_i を構成するノードとエッジの数をほぼ均一に分割できる．つまり地図の場合，地域的な粗密性などによる原地図データの偏りに各 G_i の大きさが影響されない．

[2] 各部分領域 G_i の境界抽出

各 G_i のエッジに注目し，エッジの両端ノードの所属母点が異なるエッジを抽出する．抽出したエッジを**境界エッジ**といい，境界エッジの両端ノードを**境界ノード**という（図 4 の白丸）．図 4 で，b_1^1〜b_5^1 が G_1 内の境界ノードを示す．

[3] 部分領域 G_i 内での経路スカイライン抽出

各 G_i 内において，G_i のすべての境界ノードから G_i 内の (境界ノードを含む) すべてのノードまでの経路スカイラインの抽出を行う．つまり G_i 内で，各境界ノード b_j^i を始点ノードとして隣接するノードへと順次展開し，G_i 内のすべてのノード v_i への経路スカイライン (図 4 実線) と境界ノード間経路スカイライン (図 4 点線) を抽出する．これにより，各 G_i で $G(V,E,W)$ 全域で経路スカイラインになりえない経路はすべて除去し，経路スカイラインを構成する部分経路の候補集合を求めている．つまりこの処理は，$G(V,E,W)$ 上の探査空間を予め刈込む処理に相当する．特に，経路スカイラインは部分領域の境界エッジ (境界ノード) のいずれかを必ず通過するため，各 G_i 内で予め境界ノード間の部分経路スカイラインを候補として求めておく．

v_s から v_t まで部分経路スカイラインを延長して生成した経路は，必ずしも 2 地点間の経路スカイラインになるとは限らない．そのため上記 [3] においても，また，次節 4.3 の探査実行処理においても，経路の支配関係の調査が必要である．これについては，4.4 節で探査停止基準と探査過程のモニタリングについて述べる．

4.3　探査実行処理

出発地点 v_s，目的地点 v_t と利用者選好ベクトル Π(式 (3)) を指定する．ここでは v_s, v_t は同じ部分領域に含まれないものとする．v_s から各部分領域の境界エッジを跨ぎながら，部分経路スカイラインを順次連結して，全体の経路スカイラインを求める．表 1 に，ノード，境界，経路についてのデータ構造を示す．

探査実行処理は次の 3 段階 [1]，[2]，[3] からなる．

[1：v_s から境界までの経路スカイラインの選択]　既に前処理で求められている，v_s を含む部分領域 G_{i_s} の各境界ノード ($b_j^{i_s}$) までの経路スカイラインを選択する [Algorithm 1]．Algorithm 1 で，v_s が保持する境界までの経路スカイライン (partRoute) の数だけ繰返し，境界までの経路スカイラインを生成する [1 行目]．partRoute の各コストと経路を構成するノード群を格納している [2-4 行目]．その後 5 行目以降で，境界リスト (boundaryList) から波頭ノード (route2.dstNode) と一致する境界を探し，その境界が持つ各コストと他方の境界ノード (nodeLeft) を格納する．次に nodeLeft の所属母点を判定する事で波頭ノードの探査領域を flag にセットする．この flag を確認する事で，どの領域を探査しているかが分かる．

Algorithm 1: Exploration in the v_s area

```
Input: arrayList(Boundary)boundaryList
 1: For ( i=0;  i<partRoute.size();  i++ ){
 2:  Route route2 = (Route)route.clone();
 3:  route2.setCost(partRoute.get(i).getCost1(),...);
 4:  route2.insertNode(partRoute.get(i).getNodeList);
 5:   For ( j=0;  j<boundaryList.size();  j++ ){
 6:    If (route2.dstNode == boundaryList.get(j).nodeRight()){
 7:     route2.setCost(boundaryList.get(j).getCost1(),...);
 8:     route2.insertNode(boundaryList.get(j).nodeLeft);
 9:     route2.setflag(); } } }
```

Algorithm 2−1: Exploration in intermediate areas

```
Input: arrayList(Boundary)boundaryList, Route route
 1: For ( i=0;  i<boundaryList.size();  i++ ){
 2:  If ( route.dstNode == boundaryList.get(i).nodeRight()){
 3:   If ( route.dstNode.MotherID !=
                          boundaryList.get(i).leftMotherID()){
 4:    Route route2 = (Route)route.clone();
 5:    route2.setCost(boundaryList.get(i).getCost1(),...);
 6:     If ( route2.dstNode.routeTemp.size() !=0 &&
                                route2.dstNode != goalnode ){
          // Halt condition exploration Ⅲ
 7:    } If ( route2's not dominated ){
 8:      route2.insertNode(boundaryList.get(i).nodeLeft());
 9:      route2.setflag(); } } } }
```

[2：v_s, v_t を含まない中間領域の探査]　中間領域内の G_{i_s} に隣接する部分領域 G_{i_m} 内の境界ノード $b_k^{i_s}$ から，他の全ての境界ノード $b_l^i, (k \neq l)$ までの経路スカイラインを辿り展開する．しかし注意すべき点として，展開した境界ノードが更に境界となっている場合がある．その連続で境界を渡る処理を Algorithm 2-1 に示す．Algorithm 2-1 で境界リスト (boundaryList) から探査途中経路 (route) の波頭ノード (route.dstNode) と一致する境界を探している [1-3 行目]．一致する境界が見つかると，その境界が持つ各コストを格納する [4-5 行目]．6 行目は 4.4 節で述べる，探査停止基準 Ⅲ の処理であり，支配関係の調査を行う．その結果 route2 が支配されていなければ，7 行目以降で nodeLeft と flag を格納する．

上記で，連続で境界エッジを渡る処理を行わなかった場合，つまり展開した境界ノードが，他の部分領域への境界になっていない場合である．これについては，部分領域内の経路スカイラインを順次展開する [Algorithm 2-2]．Algorithm 2-2 で，各部分領域内の経路スカイラインを格納したリスト (partRSList) から，route.dstNode と一致する経路スカイラインを探している [1-2 行目]．つまり，各部分領域内の経路スカイラインを仮想エッジとみなし順次連結し延長していく．一致すれば 3-5 行目で，その経路の持つ各コストと経路を構成するノード群を格納する．6 行目は上で述べた支配関係の調査であり，その後 7 行目で dstNode が目的地点ノードに到達していれば次の処理に移る．もし route2 が支配されていなければ，Algorithm 2-1 の境界を探す処理を実行する．

これらの処理を繰返し探査経路を延長していき，v_t を含む領域 G_{i_t} の境界ノードまで処理を進める．これらの処理において

Algorithm 2−2: Exploration in intermediate areas

```
Input: arrayList(Route)partRSList, Route route
 1: For ( i=0;  i<partRSList.size();  i++ ){
 2:  If ( route.dstNode == partRSList.get(i).srcNode()){
 3:   Route route2 = (Route)route.clone();
 4:   route2.setCost(partRSList.get(i).getCost1(),…);
 5:   route2.insertNode(partRSList.get(i).getNodeList);
 6:    If ( route2.dstNode.routeTemp.size() !=0 &&
                                  route2.dstNode != goalnode ){
         // Halt condition exploration Ⅲ
 7:    } If ( route2.dstNode == goalnode ){ continue; }
 8:      If ( route2's not dominated ){
 9:        // Algorithm 2−1
10:    } } }
```

も，4.4 節で述べる探査過程をモニタリングしながら，各種探査停止基準を用いて支配関係を調査しながら処理を行う．特に，境界間経路スカイラインの連結に際しては，前処理の段階で境界ノード間を結ぶ経路スカイラインのみが抽出されているため，境界ノードのみにおいて支配関係を調べればよい．

[3：境界から v_t までの経路スカイラインの選択]　v_t を含む部分領域 G_{i_t} の境界ノードから，前処理で既に得られている v_t までの経路スカイラインを選択する．ここまで至って，探査実行処理での探査過程において，v_s から v_t へ到達した経路がなければ，到達したこの経路が (v_s, v_t) 間の経路スカイラインの最初の候補となり，支配関係調査のための基準となる．更に，この時点で，支配されていない経路が存在すれば探査を引き続き続行する．つまり，探査が途中の経路は，他の探査途中の経路や既に v_t に到達した経路に支配されなければ，v_t に到達するまで経路探査を続行する．

4.4　探査過程のモニタリング

4.4.1　探査停止基準と探査停止処理

前処理と探査実行処理の途中で生成される経路については，必ずしも経路スカイラインとは限らない．そのため，経路の波頭ノード間で支配関係の調査が必要である．これらの処理はグラフの展開であるため，支配されると判断されれば展開を停止し，探査を終了する必要がある．そのためここでは，3.2 節の 3 つの性質に基づき，以下の 3 種類の探査停止基準と刈込み処理について述べる．

- **探査停止基準 I**：出発地点から展開し目的地点へ到達した経路集合と，新たに目的地点へ到達した経路との支配関係の調査と刈込み [Algorithm 3].

Algorithm 3 で，展開途中経路の波頭 (Route.dstNode) が目的地点 (goalnode) へ到達したか否かの判定である [1 行目]．到達済経路集合 (sroute) が空の場合に route を sroute に格納している [2 行目]．到達経路が存在し，それぞれの経路に対し支配関係の調査 (式 (5) の関係を調査) を行っている [3-4 行目]．その結果，route が sroute 内の経路に支配されていれば route の探査を停止できる．一方，route が sroute 内の経路を支配していれば，route を sroute に追加し，同時に sroute 内の支配している経路を全て削除する．

- **探査停止基準 II**：目的地点へ到達した経路集合内の各経路と探査途中経路間の支配関係の調査と刈込み [Algorithm 4].

Algorithm 4 で，route.dstNode が goalnode へ到達していない場合に sroute が存在する時，sroute 内の全経路に対し route との支配関係の調査を行う [1-2 行目]．route が sroute 内のすべての経路に支配されていなければ，探査を続行すし，支配されていれば探査を停止する．

Algorithm 3: Termination condition in exploration Ⅰ

```
Input: arrayList(Route) sroute, Route route
 1: If ( route.dstNode == goalnode ){
 2:  If ( sroute.isEmpty() ){ sroute.add( route );
 3:   }Else {  For ( i=0;  i<sroute.size();  i++ ){
 4:     dominanceCheck ( route, sroute.get(i) );  } } }
```

Algorithm 4: Termination condition in exploration Ⅱ

```
Input: arrayList(Route) sroute, Route route
 1: If ( route.dstNode != goalnode ){
 2:  If ( ! sroute.isEmpty() ){ For ( i=0;  i<sroute.size();  i++ ){
 3:     dominanceCheck ( route, sroute.get(i) );  } } }
```

Algorithm 5: Termination condition in exploration Ⅲ

```
Input: arrayList(Route) sroute, Route route
 1: If ( route.dstNode.routeTemp.size() !=0 &&
                                  route.dstNode != goalnode ){
 2:  For ( i=0;  i<route.dstNode.routeTemp.size();  i++ ){
 3:   dominanceCheck ( route,
                          route.dstNode.routeTemp.get(i) );  } }
```

- **探査停止基準 III**：複数の探査途中経路が同一ノードへ到達した場合の支配関係の調査と刈込み [Algorithm 5].

Algorithm 5 で，route.dstNode へ到達した経路スカイライン集合 (routeTemp) が存在し，かつ goalnode ではない場合の判定処理である [1 行目]．route と routeTemp 内の全経路との支配関係の調査を行う [2-3 行目]．route が routeTemp 内の経路に支配されていれば route を刈込む事ができる．一方で，route が routeTemp 内の経路を支配していれば，route を routeTemp に追加し，同時に routeTemp 内の支配している経路を routeTemp から全て削除する．これにより，出発地点から dstNode までの，2 点間に対する経路スカイラインが routeTemp に残る．探査の過程において展開された各ノードは，routeTemp 内に出発地点からの経路スカイラインを保持している．そのため，支配関係の調査による routeTemp の更新は展開される毎に行う．

4.4.2　経路属性空間と探査

探査過程の例を前節の探査停止基準 I，II，III を用いたモニタリングについて述べる．

経路コスト (式 (1)) に対して属性値をプロットする d 次元の経路属性空間 (RAS) を考える．ここに展開途中に生成される経路の d 次元経路コストを RAS にプロットし，支配関係を調査して各展開途中の経路の続行と停止を判定する．

図 5(i)〜(iii) に，属性数 $d = 2$（横軸 : 距離，縦軸 : 価格）の場合を例として RAS の利用法を示す．また，図 6(i)〜(iii) は対応するグラフの展開過程である．簡単のため，経路コストと経路を同じ記号で表す．

まず，図 6(i) で，出発地点ノード v_s から隣接するノードへ展開し（経路 a, b, c）経路コストベクトル a, b, c として RAS にプロットする（図 5(i)）．

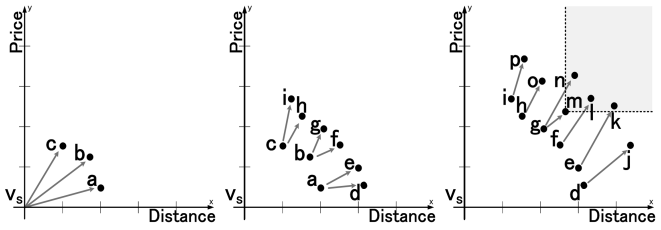

図 5　経路属性空間 (RAS) 上での展開途中の経路コストの動き

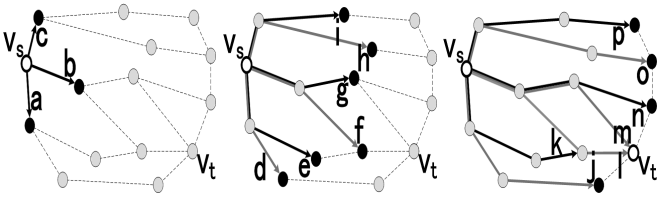

図 6　出発地点 v_s，目的地点 v_t に対する $G(V,E,W)$ の展開例

次に，経路 a,b,c の波頭ノードから，それぞれ経路 d,e，経路 f,g，経路 h,i へとそれぞれ展開する．そして，v_s から生成された各経路に対する経路コスを d,e,f,g,h,i としてプロットする (図 5(ii))．残りの 2 経路 b,c についても同様に展開し，点 f,g,h,i としてプロットする．

このように，出発地点ノードからの展開された経路をステップ毎にすべて RAS にプロットすることにより，探査過程をモニタリングする．

RAS 上にプロットされた点は以下の特徴を持つ．

- RAS 上の点は，v_s から v_t への探査途中に生成されすべての経路の経路コストを示す．
- G(V,E,W) のエッジコストが正であるため，各点は探査の進行に従って RAS 上で右上方向に移動する．
- G(V,E,W) の接続関係に依存して，RAS 上の 1 つの点 a が複数の点 d,e に分化することもある（波頭ノードが交差点などの場合）．

更に，グラフ G(V,E,W) 上で展開を続行すると，目的地点ノードへ到達する経路が出現する．この場合 RAS 上の点 m が目的地点ノード v_t に到達したことを示している (図 5(iii))．この時，経路 m は，2 点間 (v_s,v_t) に対する経路スカイラインの候補となる．このように，目的地点ノード v_t へ到達する経路が求められると，経路 m は RAS 上に永続的な支配領域を与える (図 5(iii) 点線右上領域)．この永続的な支配領域を支配関係の基準として，探査途中の各経路に対して，経路 m との支配関係を調べることができる．

図 5(iii) の場合は，点 k,n が表す展開途中の 2 つの経路が，経路 m に支配されていることを示し，[探査停止基準 II] によって，これらの経路の探査を打ち切ることができる．また，l に関しては，目的地点ノード v_t へ到達したが，他の到達経路 m により支配されていることを示している．l は [探査停止基準 I] により探査を停止する．更に，(図 6 の (ii)) の f と (図 6(iii)) の k が同一ノードへ到達している．この時，v_s から到達ノードまでの 2 つの経路間において，[探査停止基準 III] による支配関係を調査する．その結果，支配された経路の探査を打ち切ることができる．一方，支配されなかった残りの探査途中の経路 j,o,p は探査を続行し，目的地点ノード v_t に到達するか，途中で支配され探査を打ち切ることのない限り探査を続行する．最終的に支配することも支配されることもなく，目的地点ノード v_t に到達した経路の集合が，2 点間 (v_s,v_t) に対する経路スカイラインとなる．

4.5　提案手法の特徴

提案手法の特徴を以下に示す．

- **部分経路のパレート最適性を用いた刈込み**：　経路スカイラインの任意の部分経路がパレート最適である点に着目している．このため，分割された経路を順次連結し延長し繋げて，経路スカイラインを構成的に求めている．但し，3. 2[注] に基づき，探査途中で支配関係を調査している．
- **2 段階構成の探査**：　経路スカイラインを抽出するには，出発地点と目的地点に対して探査処理が必要である．提案アルゴリズムでは，全体グラフ上の経路探査を複数の小規模グラフ上で行って（前処理），それにより得られた結果を対象に 2 度目の探査（探査実行処理）を行っている．これにより，探査処理を分散化させ，効率的な経路探査を行っている．
- **境界での処理に帰着**：　グラフを予め分割して部分領域毎に経路探査している．前処理で各部分領域内の処理は終えているため，探査実行処理では，部分領域の持つ境界での処理のみを実行するだけでよく，部分領域内探査を省略できる．

5.　提案手法の評価

5.1　システム環境

提案手法を数値的に評価するために各要素についての数値シミュレーションを行う．対象とする地図データとして国土地理院数値地図 2,500(空間データ基盤，1/2,500 縮尺) を利用する．実験環境は Windows7Pro，Core i7 3.40GHz，メモリ 4.0GB である．原地図データを読み込み，前処理と利用者が出発地点ノード v_s，目的地点ノード v_t，各次元に対する利用者の選好 Π を指定してから，最終的に (v_s,v_t) 間の経路スカイラインを返すまでの探査実行処理について測定した．また，グラフ G の分割における母点選択確率は 1/2 から 1/64 まで変化させた．

比較対象として，Kriegel ら [7] の参照点埋込み法を用いた探査アルゴリズム (ARSC) を用いる．

5.2　空間分割に関する評価

生成確率と生成時間：NV 分割と部分領域内探査を含む前処理の生成時間の計測を行った．図 7 に属性数 2〜5 の，生成時間に関する母点選択確率の影響の評価として，生成時間 (m/sec) の結果を示す．

図 7 より，属性数 2〜5 に対し一つの部分領域の拡大に比例し，生成時間が増加していることが分かる．最大の生成時間は，属性数 5 の確率が 1/32 のときの約 12,000 ミリ秒であった．これは母点選択確率の変化により部分領域が拡大され，それぞれの部分領域内におけるグラフ展開に対し影響を及ぼしていることが確認できる．

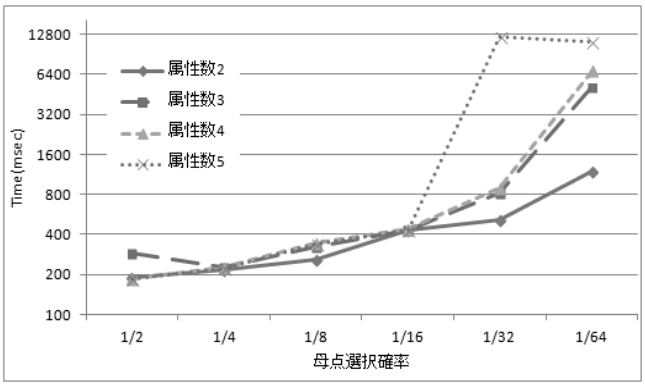

図 7　母点選択確率と各属性数に対する生成時間 (m/sec)

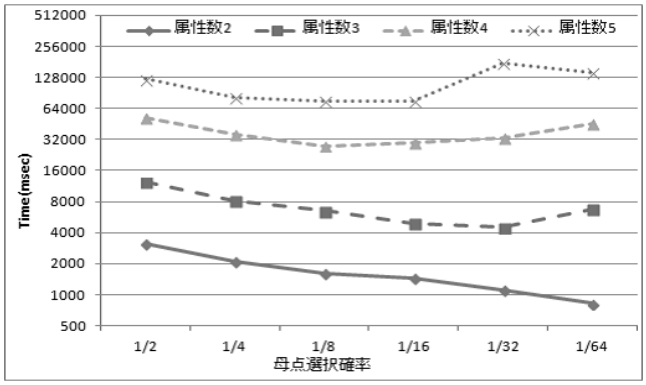

図 8　母点選択確率と各属性数に対する探査実行時間 (m/sec)

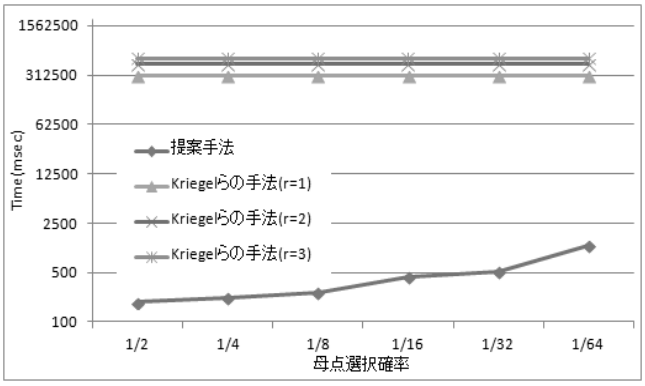

図 9　属性数 $d = 2$ に対する生成時間の比較 (提案手法 vs ARSC)

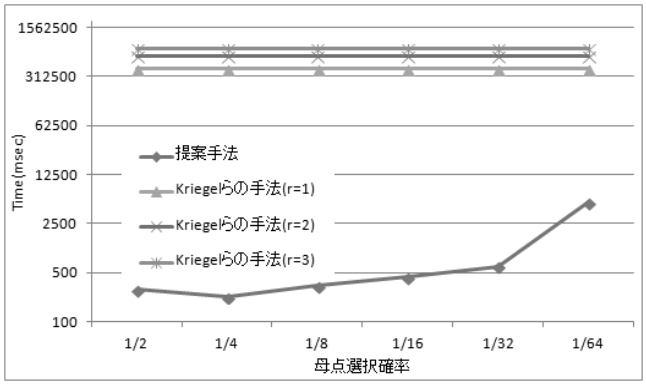

図 10　属性数 $d = 3$ に対する生成時間の比較 (提案手法 vs ARSC)

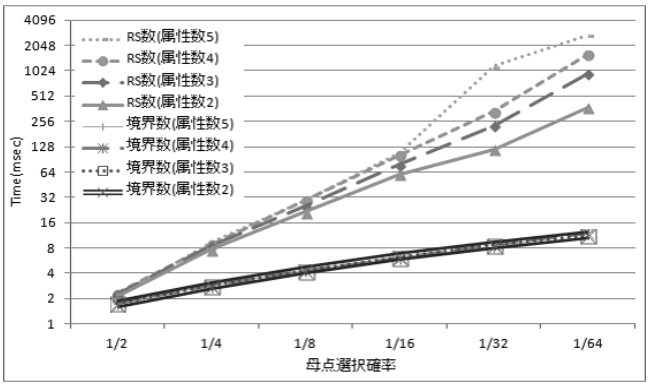

図 11　各属性数に対する部分領域内データ

各部分領域のデータ：前処理により生成される各部分領域内の経路スカイライン数と境界数の計測を行った．母点選択確率の影響による，部分領域内への影響の評価として，確率を 1/2〜1/64 まで変化させた各計測結果を図 11 に示す.

図 11 からは，各部分領域内の境界数と境界間経路スカイライン数 (RS 数) は，部分領域数の減少 (母点選択確率の減少) と共に，増加傾向がある．つまり，属性数が変化しても，部分領域内の境界数と境界間経路スカイライン数は大きく変化することはないことが確認できる．また母点選択確率が小さくなると，境界数と境界間 RS 数共に，一定数に収束し，境界数に関しては属性が変化してもほぼ同じになることがわかる.

5.3　生成と探査実行時間

比較対象の Kriegel らの ARSC では，参照点の数を $r = 1$〜3 とし，探査実行時に目的地点までの距離見積もりを用いて経路の展開を停止させている．なお各次元に対する利用者の選好 Π は出発地点ノード v_s と目的地点ノード v_t を同時に与え，解を求めている．また ARSC では，属性数 2〜3 に対する，生成時間・探査実行時間について比較した．生成時間に関する計測結果を図 9 と図 10 に示し，探査実行時間に関する計測結果を図 12 と図 13 に示す.

生成時間：Kriegel らの手法は，原地図からデータを読み込み参照点を設定する．各参照点からグラフ上の全ノードに対し，属性毎の最低コスト値を算出する．ここまでの時間を Kriegel らの手法における生成時間とし提案手法の生成時間との比較を行う．図 9 と図 10 より，提案手法の生成時間は両属性ともに，母点選択確率に依存せず優れていることがわかる．これは，Kriegel らの手法では与えられたグラフの全ノードに対し参照点から探査を行っているため，参照点の数とノード数，さらに属性数に比例し計算コストが増加していることに由来する．一方，提案手法は全領域を各部分領域に分割し，その各部分領域内において演算しているため，Kriegel らの手法と比較して計算コストを削減できている.

探査実行時間：両手法とも，利用者が出発地点ノード v_s，目的地点ノード v_t，各属性に対する利用者選好 Π を与えてから，その 2 地点間の経路スカイラインを返すまでの時間を，探査実行時間とし比較した．図 12 と図 13 より，提案手法の探査実行

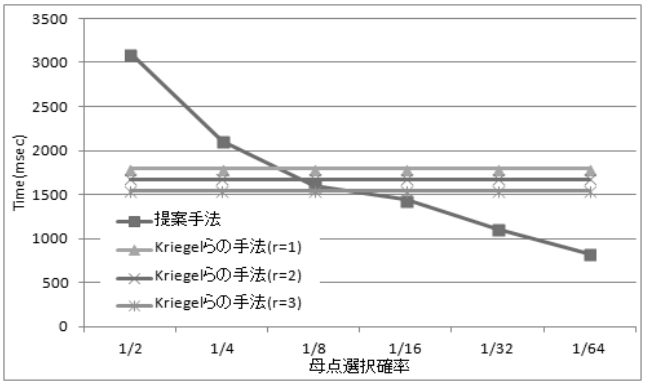

図 12　属性数 $d = 2$ に対する探査実行時間の比較 (提案手法 vs ARSC)

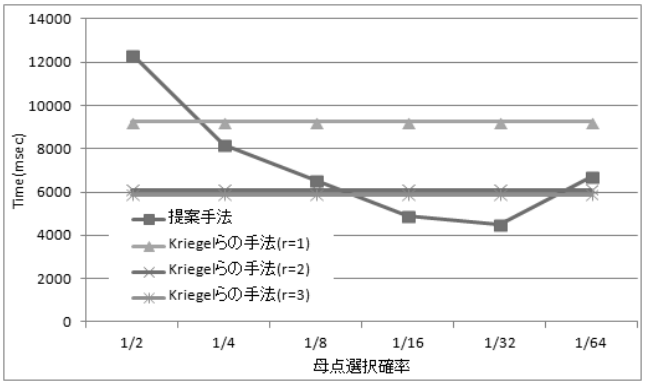

図 13　属性数 $d = 3$ に対する探査実行時間の比較 (提案手法 vs ARSC)

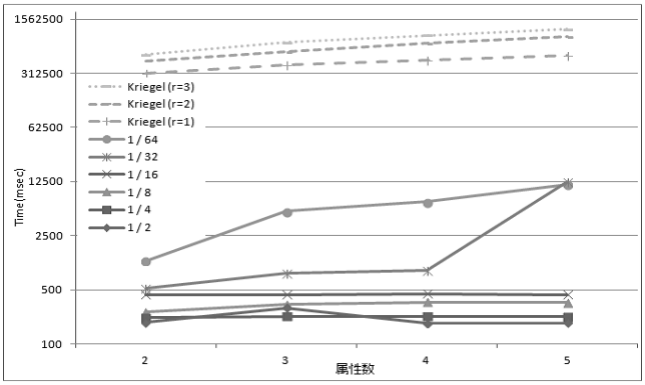

図 14　属性数 d の変化に対する生成時間の比較 (提案手法 vs ARSC)

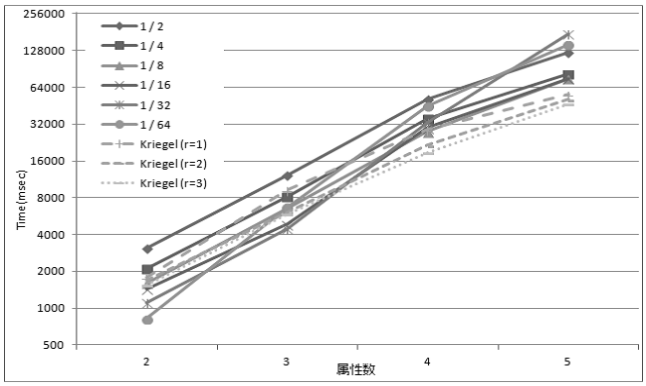

図 15　属性数 d の変化に対する探査実行時間の比較 (提案手法 vs ARSC)

時間は，母点選択確率の影響を受け変化しているが，属性数 2 に関しては確率 1/8 以下で，属性数 3 に関しては確率 1/16 と 132 で優れていることがわかる．これは，探査開始が提案手法の方が早いためであり，提案手法の有効性が確認できる．これは，Kriegel らの手法では，利用者が (v_s, v_t, Π) を与えない限り，参照点を利用した各属性毎の目的地点ノードまでの推定値演算ができないためである．これらより，提案手法は中距離〜長距離における探査では優位であり，Kriegel らの手法では近距離の探査で優れていることが確認できる．

6. おわりに

本稿では，空間分割を用いた経路スカイライン抽出アルゴリズムを提案した．提案アルゴリズムでは，前処理と探査実行処理の 2 段階から構成され，予め経路スカイラインとなる可能性のある解候補経路のみからなるグラフを作成しておくことで，経路スカイライン抽出の効率化を図り，更に経路間の支配関係によって探査打ち切りを行って，多属性グラフ上の経路探査を従来の手法より効率的に実現することができる．数値シミュレーションにより，提案手法の優位性を示した．今後の課題としては，経路探査のさらなる効率化が挙げられる．

文　献

[1] S. Börzsönyil, D. Kossmann, K. Stocker, “The Skyline Operator”, Proceedings of the 17th IEEE International Conference on Data Engineering (ICDE), pp.421–430 (2001)

[2] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, A. E. Abbadi, “Parallelizing Skyline Queries for Scalable Distribution”, Proceedings of the Extending Database Technology (EDBT), pp.112–130 (2006)

[3] S. Wang, B. C. Ooi, A. K. H. Tung, L. Xu, “Efficient Skyline Query Processing on Peer-to-Peer Networks”, Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE), pp.1126–1135 (2007)

[4] A. Vlachou, C. Doulkeridis, Y. Kotidis, “Angle-based Space Partitioning for Efficient Parallel Skyline Computation”, Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp.227–238 (2008)

[5] H. Kohler, J. Yang, X. Zhou, “Efficient Parallel Skyline Processing using Hyperplane Projections”, Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, pp.85–96 (2011)

[6] M. Sharifzadeh, C. Shahabi, “The Spatial Skyline Queries”, Proceedings of the 32nd International Conference on Very Large Data Bases(VLDB), pp.751–762 (2006)

[7] H.-P. Kriegel, M. Renz, M. Schubert, “Route Skyline Queries: A Multi-Preference Path Planning Approach”, Proceedings of the 26th IEEE International Conference on Data Engineering (ICDE), pp.261–272 (2010)

[8] F. Graf, H.-P. Kriegel, M. Renz, M. Schubert, “PAROS: Pareto Optimal Route Selection”, Proceedings of the ACM SIGMOD International Conference on Management of Data, pp.1199–1202 (2010)

[9] F. Graf, M. Renz, H.-P. Kriegel, M. Schubert, “MARiO: Multi-Attribute Routing in Open Street Map”,Proceedings of the Lecture Notes in Computer Science, Vol.6849, pp.486–490 (2011)

[10] M.: Erwig, The Graph Voronoi Diagram with Applications, Networks, Vol.36, pp.156–163 (2000)

[11] 国土地理院：数値地図（空間データ基盤）. http://sdf.gsi.go.jp/