多階層のカテゴリ分類を用いたスカイライン経路検索について

佐々木勇和† 石川 佳治††,†††

† 名古屋大学未来社会創造機構
†† 名古屋大学大学院情報科学研究科
††† 国立情報学研究所
E-mail: †yuya@db.ss.is.nagoya-u.ac.jp, ††ishikawa@is.nagoya-u.ac.jp

あらまし OSR(Optimal Sequenced Route) 検索は，指定されたカテゴリの PoI(Point of Interest) を順序通りに通過する最短経路を求める検索である．OSR 検索では，PoI のカテゴリ分類を一階層と想定しているが，多階層になっていることが一般的である．多階層のカテゴリ分類を用いて，より詳細なカテゴリを指定した場合，経路長が非常に長くなり，ユーザにとって検索結果が好ましいとは限らない．そこで，本研究では，OSR 検索にカテゴリの適合度と経路長を指標としたスカイライン検索の考え方を取り入れた SSR(Skyline Sequenced Route) 検索を提案し，効率的な計算アルゴリズムを開発する． 実データを用いたシミュレーション実験において，提案アルゴリズムは，OSR 検索のアルゴリズムを用いた方法に比べ，計算時間を大きく削減できることを確認した．

キーワード 経路探索，スカイライン検索，グラフ

1. はじめに

近年，グラフに関する研究は非常に高い注目を集めている．グラフ上における経路検索は，様々な研究分野および産業界において最も重要な問合せの一つである．本研究では，経路検索問合せのひとつである OSR(Optimal Sequenced Route) 検索に着目する．OSR 検索は，NN(Nearest Neighbor) 検索 [4], [7], [8] を拡張した検索である．NN 検索は，始点から最も近い PoI(Point of Interest) までの経路を検索するのに対して，OSR 検索は，複数の PoI を経由する最短経路を検索する．このとき，OSR 検索では，順序付けしたカテゴリ集合を指定し，そのカテゴリの PoI を順序通りに通過する経路を問合せ結果の対象とする．OSR 検索は，旅行経路の決定などの状況に有効な問合せである．

例 1 図 *1* は，道路と *PoI* からなるグラフであり，小点が交差点，三角がユーザの現在地 *(s)*，丸がレストラン $(p_1, p_2, p_6, p_9, p_{11})$，四角がカフェ $(p_3, p_4, p_8, p_{10}, p_{13})$，菱形が映画館 (p_5, p_7, p_{12})，および線分が道路を表している．現在地を始点として，{ レストラン，映画館，カフェ} を順に訪問したいという要求に対して，*OSR* 検索では，(s, p_6, p_7, p_8) を順に巡る $R1$ を出力する．

OSR 検索では，カテゴリの分類は一階層と想定しているが，実際には，図 2 のように木構造のような多階層であることが一般的である．レストランであれば，和食，中華，イタリアンなどの分類があり，さらに，和食は，寿司などに分類され，最終的には店舗の名前が最下層にくる．OSR 検索においても，より詳細なカテゴリを指定することは可能である．先ほどの例 1 において，レストランを和食，カフェをブックカフェなどと指定することにより，それに適合した最短路を求めることができる．しかし，詳細なカテゴリを指定すると，経路長が非常に長くなってしまう可能性があり，問合せ結果がユーザにとって好ましいとは限らない．また，ユーザは，どの PoI の影響により，経路長が長くなったかを問合せ結果から判断できないため，ユーザ自身がカテゴリの指定条件を変更しても，経路長を効率的に短くすることは難しい．

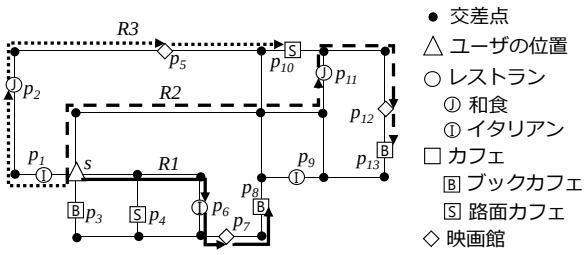


図 1: 道路

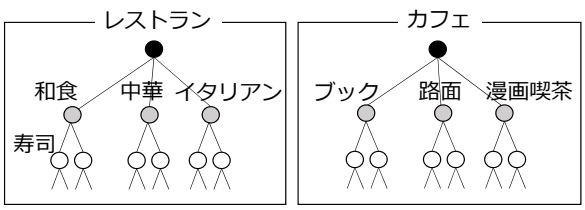


図 2: 木構造のカテゴリ分類

例 2 図 *1* において，レストランをより詳細なカテゴリに分類すると，和食 (p_2, p_{11}) とイタリアン (p_1, p_6, p_9) となり，カフェをより詳細なカテゴリに分類すると，ブックカフェ (p_3, p_8, p_{13}) と路面カフェ (p_4, p_{10}) となる．*OSR* 検索により，始点 s から始まる経路のうち，{ 和食，映画館，ブックカフェ} の順に巡る最短路 $R2 = (s, p_{11}, p_{12}, p_{13})$ を求めることができる．

{ レストラン，映画館，カフェ} を指定した例 1 に比べ，より詳細なカテゴリであるため，$R2$ は $R1$ より経路長が長くなる．$R2$ より短い経路長の経路を取得する場合，和食をレスト

ラン，もしくは，ブックカフェをカフェに変更という 2 つの選択肢がある．和食をレストランに変更した場合，問合せ結果が $R1$ となり，ブックカフェをカフェに変更した場合，問合せ結果が $R3$ となる．$R1$ は，$R2$ に比べて，経路長がかなり短くなっているが，$R3$ の経路長は，$R2$ とそれほど大きな差はない．つまり，この例では，和食を指定したことが原因となって，経路長が長くなっている．しかし，検索結果の $R2$ からのみでは判断できず，ユーザ自身が様々なカテゴリの組合せで OSR 検索を実行するのは効率的とはいえない．

そこで，本研究では，指定したカテゴリと経路上の PoI の適合度と経路長を指標とした *Skyline Sequenced Route (SSR)* 検索を提案する．SSR 検索では，ユーザが指定したカテゴリにできるだけ合っている経路，および経路長ができるだけ短い経路の集合を検索結果とする．これにより，様々なカテゴリの組合せの OSR 検索を実行せずに，一度にユーザにとって有益な全ての経路を出力可能である．本研究におけるカテゴリの適合度は，経路上の PoI が指定されたカテゴリに比べ，何階層上位かを用いる．例えば，カテゴリとして { 寿司，ブックカフェ } を指定した場合において，経路上の PoI がそれぞれ { 和食，カフェ}，{ レストラン，ブックカフェ}，および { 寿司，ブックカフェ} ならば，カテゴリ適合度はそれぞれ，2，2，および 0 となる．カテゴリ適合度と経路長の組合せを経路の指標として，他の問合せ結果の経路よりカテゴリの適合度および経路長のどちらかが小さければ（スカイライン検索における支配されていなければ），ユーザにとって有益であると考え，その経路は SSR 検索の問合せ結果に含まれる．

例 3　例 *2* の問合せ条件における，*SSR* 検索の問合せ結果は，$R1$ と $R2$ となる．$R3$ は，$R2$ より経路長が長い，かつカテゴリの適合度が同じ（どちらも *1*）であるため，問合せ結果には含まれない．

この SSR 検索を実現する最も簡単な方法は，全てのカテゴリの組合せに対して，OSR 検索を実行することである．しかし，全ての組合せを計算する場合，無駄な検索が多く発生し，計算時間が非常に膨大になってしまう．そのため，まず，カテゴリの組合せの計算順を工夫することで，SSR 検索の計算量を減らすアルゴリズムを提案する．アルゴリズムとして，OSR 検索の計算量の削減を目的とする方法 *OSR-based Algorithm for Reduction Calculation Cost (OSR-RCC 法)* と，OSR 検索の回数の削減を目的とする *OSR-based Algorithm for Reduction Number of Times (OSR-RNT 法)*）を提案する．OSR-RCC 法では，それぞれの OSR 検索の経路長の上限値を求めていき，一度の OSR 検索の計算量を削減する．一方で，OSR-RNT 法では，不必要な OSR 検索をスキップできるように設計する．これらの方法により，全ての組合わせに対して，単純に OSR 検索を実行する場合に比べ，計算時間が大きく減少するが，OSR 検索を複数回実行しなければいけないため，それほど効率的とはいえない．そこで，一度の問合せ処理で SSR 検索の結果を求める，*Bulk Skyline Sequenced Route Algorithm (BSSR 法)* を提案する．BSSR 法では，まず，SSR 検索の経路長の上限値を最近傍検索を基とした方法で求める．その後，その上限値以下の経路を，経路に含まれる PoI 数，カテゴリの適合度，および経路長を考慮した順で検索していく．このとき，上限値をカテゴリの適合度毎に更新していくことで，計算範囲を削減する．

提案手法の評価として，実際の道路図と PoI を用いたシミュレーション実験を行なう．実験結果より，提案手法は，単純な OSR 検索を繰り返す手法に比べて，実行時間を大きく削減できることを確認した．特に，BSSR 法は，他の提案手法に比べても，半分程度の実行時間で，SSR 検索を実現できる．

論文の構成は以下の通りである．まず，2. にて関連研究，3. にて問題定義についてそれぞれ述べる．その後，4. にて提案手法について説明し，5. にて提案手法の性能評価を行う．最後に，6. にて本稿をまとめる．

2. 関 連 研 究

本章では，まず OSR 検索，およびその他の類似した問合せについて述べる．その後，グラフ上のスカイライン経路検索について説明する．

OSR 検索は，論文 [9] にて，Shrifzadeh らにより提案された．論文 [9] では，OSR 検索を効率的に解く方法として，それぞれダイクストラベース法，R-LORD 法，および PNE 法を提案している．R-LORD 法は，空間データベースを対象とし，PoI 間の距離は，ユークリッド距離としている．一方，PNE 法では，道路ネットワークを対象としており，PoI 間の距離は，グラフの枝の重みの総和としている．ダイクストラベース法は，どちらの距離にも適応可能な手法である．本研究では，グラフを対象とするため，ダイクストラベース法と PNE 法が有効である．3.2 節にて，ダイクストラベース法と PNE 法について説明する．Seherifzdeh らは，ボロノイ図を用いたインデックス技術を用いて，OSR 検索の効率化を行っている [10]．本研究では，PoI のカテゴリの種類数が非常に多いことを想定しているため，このインデックス技術では，メモリ領域を大きく使用することが予測される．そのため，このインデックス技術は用いずに，オンライン検索により実現する．OSR 検索をより一般化した問合せに，Multi-Rule Partial Sequenced Route 検索 (MRPSR 検索) [3] がある．OSR 検索では，PoI のカテゴリの順序を全て決めているが，MRPSR 検索では，一部のカテゴリの順序だけ与えている（例えば，銀行の後に，ショッピングモールと指定した場合，銀行はショッピングモールの前に訪問しなければいけないが，銀行とショッピングモールの間に他の指定されたカテゴリが含まれていても問題ない）．この論文では，空間データベースを対象にしており，グラフに関しては言及していない．その他の類似した問合せ処理に，旅行計画問合せ [6] と Keyword-aware Optimal Route (KOR) 検索 [2] がある．旅行計画問合せでは，始点，終点，PoI のカテゴリ集合（順序なし）を指定し，PoI のカテゴリを通過し，始点から終点までの最短路を検索する．KOR 検索では，枝にコストと品質，節点にキーワードが付与されているグラフを想定する．始点，終点，キーワード群，および最大コストが指定し，2 つの条件: 1) 始

点から終点までの経路上のコストは，最大コスト以下である，2) 経路は全てのキーワードを含む，を満たす品質が最大となる経路を求める．旅行計画問合せおよ KOR 検索では，終点を指定する点などが異なるため，SSR 検索とは異なる．

グラフ上のスカイライン経路検索についての研究も盛んに行われている [5], [11], [12]．この種のスカイライン経路検索では，グラフの枝に様々なコスト（移動距離，移動時間，信号機の数など）が割り当てられており，経路は複数のコストをもつ．この経路のコストを用いて，スカイライン [1] を求め，スカイライン経路としている．本研究においても，経路長とカテゴリ適合度という 2 つのコストを用いるスカイライン経路検索である．しかし，これらの研究では，指定した始点と終点間の経路検索を想定している．複数の PoI を指定する問合せを対象とするスカイライン検索は，著者らの知る限り存在せず，既存の研究では，単純に解決することはできない．

3. 問 題 定 義

本稿では，連結グラフ $G = (V, E)$ を想定する．V および $E \subseteqq V \times V$ は，それぞれ節点集合および枝集合を表す．このグラフは，道路ネットワークや，ユーザの軌跡情報から作成されると想定する．節点 $v \in V$ のうち，いくつかの節点はカテゴリをもつものとし，カテゴリをもつ節点を PoI とする．カテゴリ分類は多階層になっており，最上位のカテゴリ $(c_{\{i\}}, 1 \leqq i \leqq N)$ を根とする木構造とする．本稿では，簡単化のため，木の高さ h は全て同じものとするが，異なる高さでも問題ない．カテゴリの木構造において，深さ j $(\leqq h)$ のカテゴリ識別子を $c_{\{i_1,i_2,\cdots,i_j\}}$ とする．例えば，図 2 における，寿司，イタリアン，および漫画喫茶のカテゴリ識別子はそれぞれ $c_{\{1,1,1\}}$，$c_{\{1,3\}}$，および $c_{\{2,3\}}$ となる．また，v_i から v_j 間の枝を $e(i,j) \in E$ と表し，$e(i,j)$ は，重み $e(i,j).w$ (> 0)（節点間の距離や移動時間など）をもつものとする．本稿では，無向グラフとして説明するが，有向グラフへの拡張も容易である．

3.1 SSR 検索の定義

本章では，本稿で用いる用語を定義する．

定義 1　経路. $R = (v_0, v_1, \cdots, v_n)$ を v_0 から v_n までの経路とし，v_0 は始点，v_i $(1 \leqq i \leqq n)$ は PoI（つまり，カテゴリをもつ節点）とし，$v_i.c$ を v_i のカテゴリ識別子とする．

定義 2　シーケンス. $M = (M_1, M_2, \cdots, M_m)$ をシーケンスとよぶ．M_i $(1 \leqq i \leqq m)$ はカテゴリ識別子である，

定義 3　カテゴリ適合度. *2* つのカテゴリ $c_{\{a_1,\cdots,a_x\}}$，$c_{\{b_1,\cdots,b_y\}}$ $(h \geqq x \geqq y)$ が与えられたとき，*2* つのカテゴリの最上位のカテゴリから順に走査していき，どの階層まで同じかをカテゴリ適合度 $c(c_a, c_b)$ とし，$a_i = b_i$ $(1 \leqq i \leqq j)$ の場合，カテゴリ適合度を $h - j$ とする．また，$a_i = b_i$ $(1 \leqq i \leqq y)$（カテゴリ適合度が $h - y$）の場合，完全カテゴリ適合とよび，$a_1 \neq b_1$ の場合，カテゴリ適合しないとよぶ．

上記の定義より，以下のシーケンス経路を定義する．

定義 4　シーケンス経路. シーケンス $M = (M_1, M_2, \cdots, M_m)$ と経路 $R = (v_0, v_1, \cdots, v_n)$ が与えられ，$m = n$ かつ v_i と $M_i (1 \leqq i \leqq m)$ が適合する場合，その経路はシーケンス経路 SR である．さらに，SR のスコアとして，経路長 $l(SR)$ とカテゴリ適合度 $c(SR)$ を定義する．

$$l(SR) = \Sigma_{i=0}^{n-1} D(v_i, v_{i+1}). \quad (1)$$

$D(v_i, v_{i+1})$ は，v_i から v_{i+1} までのグラフ上の最短経路を表す．

$$c(SR) = \Sigma_{i=1}^{m} c(v_i.c, M_i). \quad (2)$$

$l(SR)$ が小さいと，経路の長さが短く，$c(SR)$ が小さいと，経路上の節点がシーケンスとより適合している．どちらの値も小さい方がよい経路となる．

図 1 において，{ 和食，映画館，ブックカフェ } が指定された場合において，$l(R1)$，$c(R1)$，$l(R2)$，$c(R2)$，$l(R3)$，および $c(R3)$ は，それぞれ，4.5，1，8.5，0，7.5，および 1 となる．$R3$ においては，p_1 と p_2 のどちらも経由するが，カテゴリ適合度がより小さくなる p_2 をカテゴリ適合度の計算に用いる．
次に，スカイライン検索 [1] と SSR 検索を定義する．

定義 5　スカイライン検索. 点 $p = (p_1, p_2, \cdots, p_x)$ と点 $q = (q_1, q_2, \cdots, q_x)$ において，$\forall p_i, q_i$ において $p_i \leqq q_i$，かつ $\exists p_i, q_i$ において $p_i < q_i$ ならば，p は q を支配しているという．スカイライン検索は，支配されていない点を全て検索する．

スカイライン検索をシーケンス経路に組込み，SSR 検索を以下の定義となる．

定義 6　SSR 検索. 始点 s，およびシーケンス M_q が与えられ，s から始まるシーケンス経路を SR_i，SR_j としたとき，$l(SR_i) \leqq l(SR_j)$ かつ $c(SR_i) < c(SR_j)$，もしくは $l(SR_i) < l(SR_j)$ かつ $c(SR_i) \leqq c(SR_j)$ であるならば，SR_i は SR_j を支配しているという．*SSR* 検索は，他のシーケンス経路に支配されていない経路をスカイライン経路とし，その集合 *SSR* を問合せ結果とする．

最後に，手法の議論のために，サブシーケンスと部分シーケンス経路を定義する．

定義 7　サブシーケンス. *2* つのシーケンス $M1 = (M1_1, M1_2, \cdots, M1_m)$，$M2 = (M2_1, M2_2, \cdots, M2_m)$ 間のカテゴリ適合度を以下の式で定義する．

$$c(M_1, M_2) = \Sigma_{i=1}^{m} c(M1_i, M2_i). \quad (3)$$

$c(M_1, M_2)$ が *0* に近いほど，*2* つのシーケンスはより似ている．また，M_1 と M_2 の要素のうち，*1* つでも適合していないなら，そのシーケンスは適合していないとする．シーケンス M のサブシーケンスを，M に適合する全てのシーケンス（M 自身も M のサブシーケンスに含まれる）とする．また，カテゴリ適合度が最も大きいサブシーケンスを M_{max} とする．

表 1: 記号の一覧

記号	意味
M	シーケンス
m	シーケンスの長さ
R	経路
SR	シーケンス経路
pSR	始点からシーケンス M_i までの部分的な SR
n	経路の PoI 数．$R.v_0$ は始点，$R.v_n$ は終点
M_q	問合せに指定されたシーケンス
s	問合せに指定された始点
M_{sub}	M のサブシーケンス
M_{max}	M と比べて最もカテゴリスコアが大きい M_{sub}
$c(a,b)$	a と b のカテゴリ適合度
$c(M_1,M_2)$	M_1 と M_2 のカテゴリ適合度

表 2: OSR 検索の処理例

	ダイクストラベース	PNE
1	(s)	(s)
2	$(p_1),(p_6),(p_9),(p_{11})$	(p_1)
3	$(p_6),(p_1,p_7),(p_1,p_5),$ $(p_9),(p_{11}),(p_1,p_{12})$	$(p_6),(p_1,p_7)$
4	$(p_6,p_7),(p_1,p_5),(p_1,p_7),$ $(p_6),(p_9),(s,p_{11}),(s,p_6,p_5)$	$(p_2),(p_6,p_7),(p_1,p_7),$
5	$\underline{(p_6,p_7,p_8)},(p_6,p_7,p_4),$ $(p_6,p_7,p_3),(p_6,p_5),(p_1,p_5),$ $(p_1,p_7),(p_1,p_{12}),$ $(p_6),(p_9),(p_{11})$	$(p_6,p_7),(p_1,p_7),$ $(p_2,p_5),(p_9)$
6	Done	$\underline{(p_6,p_7,p_8)},(p_1,p_7),$ $(p_2,p_5),(p_9),(p_6,p_{12})$

例えば，{ 寿司，ブックカフェ} というシーケンスのサブシーケンスは，{ 寿司，ブックカフェ}，{ 和食，ブックカフェ}，{ レストラン，ブックカフェ}，{ 寿司，カフェ}，{ 和食，カフェ}，および { レストラン，カフェ} となり，シーケンスとの適合度は，それぞれ 0, 1, 2, 1, 2, および 3 となる．M_{max} は，{ レストラン，カフェ} である．

定義 8 部分シーケンス経路. シーケンス $M=(M_1,M_2,\cdots,M_m)$ と経路 $R=(v_0,v_1,\cdots,v_n)$ が与えられ，$n<m$ かつ M_i $(1 \leqq i<n)$ までのシーケンス経路を，部分シーケンス経路 pSR とする．pSR のスコアは，SR のスコアの計算方法とほぼ同じであるが，M_i までのカテゴリ適合度を pSR のカテゴリ適合度する．また，pSR が次に探索すべきカテゴリは，M_{n+1} であり，$v_p.c$ が M_{n+1} と適合する場合，新たな部分シーケンス経路を $pSR+v_p$ と表記する．

例えば，$R1$ の部分シーケンス経路は，$pSR=(s,p_6,p_7)$ となる．OSR 検索や SSR 検索では，部分シーケンス経路をより長くしながら，シーケンス経路を探索していく．

テーブル 1 に本稿で用いる記号の一覧を示す．

3.2 OSR 検索アルゴリズム

本章では，論文 [9] で提案されている，既存の OSR 検索アルゴリズムであるダイクストラベース法と PNE 法を簡単に説明する．OSR 検索アルゴリズムでは，カテゴリ適合度という概念がないため，完全適合する PoI のみを探索する．これらの方法では，探索済みの経路 pSR を，経路長の昇順にソートする優先度付きキュー（初期値は始点 s）に挿入し，先頭から（キュー内の経路長が最短のものから）順に取り出して処理していく．このとき，ダイクストラベース法と PNE 法では，キューに挿入する経路が異なる．

ダイクストラベース法は，キューから取り出した経路 pSR の終点から全ての M_{n+1} に完全適合する PoI，v_p，を探索し，全ての $pSR+v_p$ をキューに挿入する．これを繰り返し，M_q と適合する経路である SR がキューの先頭にきた場合に，検索を終了する．この方法は，それぞれの PoI を探索するのに，大きな時間がかかるという欠点が存在する．本研究では，このダイクストラベース法を OSR 検索にさらに適応するよう改良して用いる．既存のダイクストラベース法では，v_n から全ての完全適合する PoI を探しているが，最短路上に適合するから PoI が存在する節点は探索する必要がない．本稿のダイクストラベース法ではこの枝刈りを取り入れる．例えば，図 1 において，始点から p_2 までの最短経路上に p_1 が存在するため，レストランと完全適合する節点を検索する際に，始点から p_2 までの経路は，キューに挿入する必要はない．これにより，検索範囲を削減することが可能である．

次に，PNE 法を説明する．PNE 法では，キューから取り出した pSR の終点から最も近い未探索の M_{n+1} に完全適合する PoI，v_p，を探索し，$pSR+v_p$ をキューに挿入する．このままでは，それぞれの経路の終点から最も近い PoI のみを探索していくため，最適解をみつけることができない．そこで，v_{n-1} からの新たな未探索の M_n に適合する PoI，v_p，を見つけ，$pSR=(v_0,\cdots,v_{n-1},v_p)$ も同時にキューに挿入することにより，局所最適解になることを防ぐ．SR を発見後，キュー内の経路の経路長が SR よりも大きくなった場合，検索を終了する．この方法は，それぞれの節点を探索する時間は短いが，探索範囲が広い場合に，繰り返しが非常に多くなってしまい，効率が低下するとった欠点が存在する．

例 4 *例 1 における，OSR 検索処理の流れを説明する．表 2 は，それぞれの手法における，キュー内の経路を表す．ダイクストラベース法は，部分シーケンス経路を全て挿入していき，PNE 法は，最短路の部分シーケンス経路とひとつ前の部分シーケンス経路のみを挿入していく．ダイクストラ法は，キュー内の経路数は多くなるが，早く終了し，一方，PNE 法は，キュー内の経路数は少ないが，何度も繰り返し，後戻りするため，終了すまでの探索回数は多い．*

これらの手法の性能は環境に依存するため，本稿では，両方の OSR 検索を用いる．

3.3 単純な SSR 法

SSR 検索を最も簡単に実行する方法は，シーケンス M_q の全てのサブシーケンスに対して，OSR 検索を実行する方法で

ある．アルゴリズム 1 に単純に SSR 検索を実行するアルゴリズムを示す．それぞれのサブシーケンスに対して OSR 検索を実行（4 行目）し，その検索結果の経路が他の経路に支配されていないなら検索結果に加え，SSR 内の経路のうち検索結果に支配されている経路を削除する（5,6 行目）．SSR 検索を単純に実行する方法では，サブシーケンス数は，最大で h^m となり，全ての OSR 検索を実行するまで大きな時間がかかる．

Algorithm 1: Naive SSR search

```
   input  : startpoint s, sequence M_q
   output: SSR
1  SSR ← φ
2  M_sub ← sub-sequence of M_q
3  for ∀M ∈ M_sub do
4      SR_osr ← OSR(s, M)
5      if SR_osr is not dominated by ∀SR ∈ SSR then
6          SSR.update( SR_osr)
```

4. 提 案 手 法

OSR 検索を複数回実行する OSR ベースの方法である，OSR-RCC 法と OSR-RNT 法，および一度の検索で SSR 検索を実行する BSSR 法を提案する．

4.1 OSR ベース法

OSR 検索を複数回実行して，SSR 検索を実現する場合，1 度の OSR 検索の計算量を削減すること，および計算回数を減らすことが重要となる．まず，1 度の OSR 検索の計算量を減らすために，OSR 問合せ結果の経路の経路長の上限値を更新しながら，SSR を求める方法を提案する．

定理 1 *SSR* に含まれる経路 SR_i と SR_j において，$c(SR_i) \leqq c(SR_j)$ の場合，$l(SR_i) \geqq l(SR_j)$ である．

証明. 背理法を用いて証明する．$c(SR_i) \leqq c(SR_j)$，$l(SR_i) < l(SR_j)$ と仮定した場合，SR_j は SR_i に支配されるため，SSR には，SR_j は存在しない．□

この定理より，SSR では，適合度がより小さい経路の経路長が必ず大きいため，この経路長を上限値として用いることができる．

定理 2 カテゴリ適合度が *0* となる SR より経路長が長い経路は，*SSR* に含まれない．

証明. 定理 1 より，自明である．□

OSR 検索は，シーケンスに完全適合する SR のうち，経路長が最短のものを出力する．そのため，OSR 検索によって求められたカテゴリ適合度が 0 の SR より，他の SSR に含まれる経路の経路長が長くなることはない．このことから，カテゴリ適合度が 0 となるシーケンスから順に求めると，OSR 検索の探索範囲を徐々に小さくすることができることがわかる．そこで，OSR-RCC 法では，M_q のサブシーケンス M_{sub} と，M_q のカテゴリ適合度を計算し，カテゴリ適合度が小さいサブシーケンスから順に OSR 検索を実行していく．

アルゴリズム 2 に OSR-RCC 法の擬似コードを示す．まず，サブシーケンスをカテゴリ適合度の昇順に並べ替え（3 行目），サブシーケンスの要求が厳しい順から OSR 検索を実行していく．OSR 検索の出力結果の経路の経路長より，次の OSR 検索の経路が長くなることはないこと用いて，一度の OSR 検索の検索範囲を上限値 ub 以下にすることが可能である．より小さい経路長が現れたら，ub をより小さくしていくこと（7,8 行目）により，探索範囲をより小さくしていく．これにより，それぞれの OSR 検索の探索時間を小さくすることができる．

Algorithm 2: OSR-RCC Algorithm

```
   input  : startpoint s, sequence M_q
   output: SSR
1  SSR ← φ
2  M_sub ← sub-sequence of M_q
3  sort M ∈ M_sub in ascending order by c(M_q, M)
4  ub ← inf
5  for ∀M ∈ M_sub do
6      SR_osr ← OSR(s, M, ub)
7      if ub ≧ l(SR_osr) then
8          ub ← l(SR_osr)
9      if SR_osr is not dominated by ∀SR ∈ SSR then
10         SSR.update( SR_osr)
```

例 3 を OSR-RCC 法を用いて処理する場合，まず，{ 和食，映画館，ブックカフェ} をシーケンスとして指定した OSR 検索により，$R2$ を検索結果として得る．次に，{ レストラン，映画館，ブックカフェ} を指定し，$R1$ を得る．次の { 和食，映画館，カフェ} では，$R1$ より短いものが検索できないため，OSR 検索を途中で終了する．最後に，{ レストラン，映画館，カフェ} を指定し，$R1$ を取得し，終了する．

上記の方法では，OSR 探索の計算時間をより小さくしながら計算できるが，OSR 検索の回数を削減することはできない．次に，計算回数を削減する OSR-RNT 法を提案する．この方法では，シーケンスのカテゴリ適合度が大きい順，つまりシーケンスの要求が緩い順に，OSR 検索を実行する．OSR 検索を実行した際に，要求するシーケンスより，カテゴリ適合度が小さい経路が問合せ結果となる可能性がある．その場合，問合せ結果となったシーケンスのサブシーケンスは，計算する必要がなく，計算回数を削減することができる．

アルゴリズム 3 に OSR-RNT 法の擬似コードを示す．まず，サブシーケンスをカテゴリ適合度の昇順に並べ替え（3 行目），サブシーケンスの要求が緩い順から OSR 検索を実行していく．また，経路長を格納していき，カテゴリ適合度に対応する上限値として利用する（7 行目および 9,10 行目）．OSR 検索の出力結果の経路のシーケンスのサブシーケンスは，計算する必要がなくなるため，サブサブシーケンス内から削除する（11,12 行目）．これにより，計算回数を削減することが可能である．し

かし，カテゴリ適合度が大きいものから順に計算するため，上限値をそこまで有効に使うことはできない．

Algorithm 3: OSR-RNT Algorithm

input : startpoint s, sequence M_q
output: SSR

```
1  SSR ← φ
2  M_sub ← sub-sequence of M_q
3  sort M ∈ M_sub in descending order by c(M_q, M)
4  for 0 ≧ i ≧ c(M, M_max) do
5      ub[i] ← inf
6  for ∀M ∈ M_sub do
7      ub ← min_{i=0}^{c(M_q,M)} ub[i]
8      SR_osr ← OSR(s, M_sub, ub)
9      if ub[c(SR_osr)] ≧ l(SR_osr) then
10         ub[c(SR_osr)] ← l(SR_osr)
11     if SR_osr covers sequences in M_sub then
12         delete the sequences from M_sub
13     if SR_osr is not dominated by ∀SR ∈ SSR then
14         SSR.update(SR_osr)
```

例 3 を OSR-RNT 法を用いて処理する場合，まず，{ レストラン，映画館，カフェ} をシーケンスとして指定し，最短経路である $R1$ を検索結果として得る．このとき，$R1$ の PoI のカテゴリは，{ レストラン，映画館，ブックカフェ} であるため，このシーケンスを M_{sub} から削除する．次に，{ 和食，映画館，カフェ} を検索する場合，$R1$ の経路長が上限値となるため，OSR 検索を途中で終了する．最後に，{ 和食，映画館，ブックカフェ} を指定し，$R2$ を取得する．

4.2 Bulk SSR 法

OSR 検索を繰り返す場合，同じ経路を何度も探索する必要があるため，一度に SSR 検索を実現できるほうが計算時間を削減できる．そこで，一度の探索で SSR 検索の問合せ結果を可能とする Bulk SSR(BSSR) 法を提案する．この方法では，基本的に，探索済み経路の終点（初期値は始点）から，適合する PoI を近い順に探索していき，探索した経路を優先度付きキューに挿入していく．探索時は，M_q の M_{max} をシーケンスとして用いることにより，SSR 検索で検索結果となりうる全ての適合する経路を探索していく．処理の効率化には，上限値を早く小さくし，不必要な経路を検索しないことが重要である．まず，簡易に上限値を求め，始点 s からの探索範囲を制限する．この上限値の計算では，始点から最近傍の適合する PoI を探索，その PoI から次に適合する PoI を探索という順に，最近傍検索を繰り返すことで求める．このとき，M_q に完全適合する PoI のみを探索対象とすることにより，カテゴリ適合度が 0 となるシーケンス経路を求める．次に，検索処理を実行しながら，上限値を改善していく．上限値を早く改善するためには，シーケンス経路をできるだけ早く探索する必要がある．そのため，幅優先探索より，深さ優先探索の方が，有効であるため，優先度付きキュー内の経路は，経路内の PoI 数 n が大きい経路を先頭にくるようにする．さらに，カテゴリ適合度が小さい経路の上限値の方が，カテゴリ適合度が大きい経路の上限値より，多くの経路の探索範囲を小さくすることができる．そのため，n が同じ場合，カテゴリ適合度の小さい方を前に挿入する．最後に，カテゴリ適合度も同じ場合は，OSR 検索と同様に，経路長が短い方を前に挿入する．

Algorithm 4: Bulk SSR search

input : startpoint s, sequence M
output: SSR

```
1  SSR ← φ
2  for 0 ≧ i ≧ c(M, M_max) do
3      ub[i] ← inf
4  ub[0] ← NN-based sequenced route length
5  priority_queue PQ ← {s}
6  while PQ is not empty do
7      pSR ← PQ.dequeue()
8      if l(pSR) > MinUB(pSR) then continue
9      while nextPoI(pSR,ub) is not null do
10         pSR ← pSR+ nextPoI(pSR,ub)
11         if l(pSR) ≧ MinUB(pSR) then break
12         if pSR is SR then
13             if ub[c(pSR)] ≧ l(pSR) then
14                 ub[c(pSR)] ← l(pSR)
15             if pSR is not dominated by ∀SR ∈ SSR then
16                 SSR.update(pSR)
17         else
18             PQ.enqueue(pSR)
```

BSSR 法の大きな利点は，同じ経路を何度も探索しない点，および部分シーケンス経路に対しても，上限値による枝刈りを行なうことができる点である．これにより，大きくキューに挿入する経路数も削減することができる．

アルゴリズム 4 に BSSR 法の擬似コードを示す．まず，最近傍検索ベースの方法により，カテゴリ適合度が 0 となるシーケンス経路の上限値を取得する（4 行目）．キューの先頭より，経路を取り出し，取り出した経路のカテゴリ適合度の経路長より，小さい上限値が存在すれば，その経路は計算しない（8 行目）．`nextPoI` は，$pSR.v_n$ からの最も近い未探索の適合する PoI を探索するが，カテゴリ適合度に対応する上限値より大きい経路しかない場合，Null を返す（9 行目）．新たに求めた経路の経路長が上限値より大きくなった場合，キューから新たに経路を取り出す（11 行目）．新たに求めた経路の経路長が上限値より小さい場合は，シーケンス経路かどうか判断（14 行目）し，シーケンス経路であれば，上限値の更新と SSR の更新を行なう (15–18 行目)．シーケンス経路でない場合は，キューに逐次挿入していく．

例 3 を BSSR 法で処理する流れを説明する．表 3 にキューと SSR 内の経路を示す．まず，最近傍検索の繰り返しにより，(p_2, p_5, p_8) というカテゴリ適合度が 0 のシーケンス経路を探索

表 3: BSSR 法の処理例

1	queue: (s) SSR: (p_2, p_5, p_8)
2	queue: $(p_2), (p_{11}), (p_1), (p_6), (p_9)$, SSR: (p_2, p_5, p_8)
3	queue: $(p_2, p_5), (p_{11}), (p_1), (p_6), (p_9)$, SSR: (p_2, p_5, p_8)
4	queue: $(p_{11}), (p_1), (p_6), (p_9)$ SSR: $(p_2, p_5, p_8), (p_2, p_5, p_{10})$
5	queue: $(p_{11}, p_{12}), (p_1), (p_6), (p_9)$ SSR: $(p_2, p_5, p_8), (p_2, p_5, p_{10})$
6	queue: $(p_1), (p_6), (p_9)$ SSR: $(p_{11}, p_{12}, p_{13}), (p_2, p_5, p_{10})$
7	queue: $(p_1, p_7), (p_1, p_5), (p_6), (p_9)$ SSR: $(p_{11}, p_{12}, p_{13}), (p_2, p_5, p_{10})$
8	queue: $(p_1, p_5), (p_6), (p_9)$ SSR: $(p_{11}, p_{12}, p_{13}), (p_1, p_7, p_8)$
9	queue: $(p_6), (p_9)$ SSR: $(p_{11}, p_{12}, p_{13}), (p_1, p_7, p_8)$
10	queue: $(p_6, p_7), (p_9)$ SSR: $(p_{11}, p_{12}, p_{13}), (p_1, p_7, p_8)$
11	queue: (p_9) SSR: $(p_{11}, p_{12}, p_{13}), (p_6, p_7, p_8)$
12	Done SSR: $(p_{11}, p_{12}, p_{13}), (p_2, p_5, p_{10}), (p_6, p_7, p_8)$

し，この経路長 9.5 を上限値とする．3 において，p_2 から，次に適合する PoI の候補として，p_5，p_7，および p_{12} があるが，p_7 と p_{12} までの経路長は，9.5 を超えるため，キューに挿入しない．これにより，無駄な探索を抑えつつ，計算量を削減している．

5. 性 能 評 価

本章では，BSSR 法（BSSR），ダイクストラベース法を OSR 検索手法として用いた OSR-RCC 法（OSR-RCC(dijk)），PNE 法を OSR 検索手法として用いた OSR-RCC 法（OSR-RCC(PNE)），ダイクストラベース法を OSR 検索手法として用いた OSR-RNT 法（OSR -RNT(dijk)），および PNE 法を OSR 検索手法として用いた OSR-RNT 法（OSR-RNT(PNE)）の 5 つの手法を評価する．3.3 節で述べた単純手法は，著しく性能が悪かったため，評価から割愛する．実験では， Intel Core i7-7400 3.40GHz，32.0GB RAM の計算機を用いた．

5.1 セットアップ

本実験では，グラフとして，カリフォルニア州の道路図（図 3）と PoI(注1) を用いた．道路図における交差点数と道路数は，それぞれ，21048，および 22830，PoI 数は 87635 である．PoI は，緯度経度情報より取得されるが，最も近い道路上に節点として追加した．PoI の各階層のカテゴリ数を 5 とし，カテゴリの高さ h をデフォルト 3 とし，1 から 6 の間で変化させた．最下層の総カテゴリ数は 5^h となり，それぞれの PoI に一様乱数

(注1)：http://www.cs.utah.edu/ lifeifei/SpatialDataset.htm

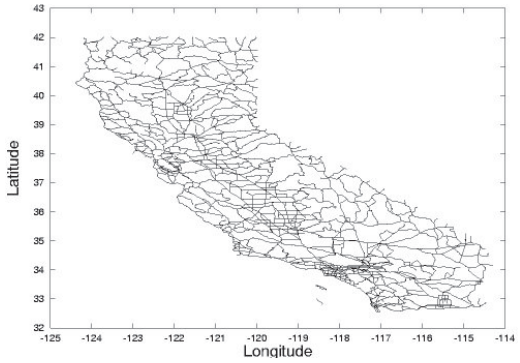
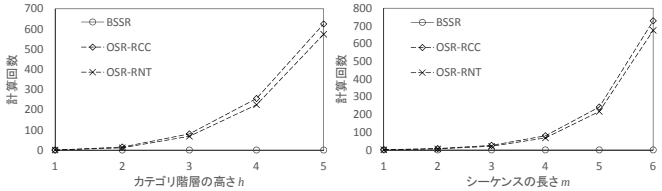
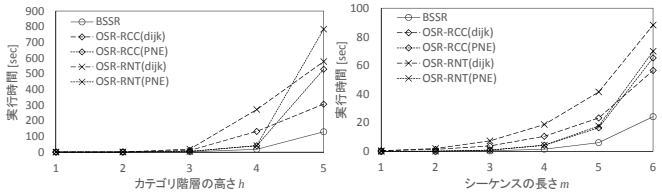


図 3: カリフォルニア道路図



(a) h の影響　(b) m の影響
図 4: OSR 検索の計算回数



(a) h の影響　(b) m の影響
図 5: 実行時間

を用いて振り分けた．SSR 検索において，ユーザが指定する始点は交差点の中からランダムで決め，シーケンスの長さ m はデフォルト 4 とし，1 から 5 の間で変化させた．指定するカテゴリは最下層のカテゴリから，最上位のカテゴリは重複しないように，ランダムで決定する．SSR 検索を 100 回実行した場合の OSR 検索の計算回数，および実行時間の平均を評価軸とする．

5.2 OSR 検索の回数

この評価では，OSR-RNT 法がどの程度計算回数を削減できているかを調べる．SSR 検索における OSR 検索の回数を図 4 に示す．BSSR 法は，一度の検索であるため，常に 1 である．また，OSR-RCC 法の計算回数は，全てのサブシーケンスの組合せに対して OSR 検索を行なうため，h^m となる．OSR-RNT 法の計算回数は，それほど削減できていないことがわかる．これは，SSR 検索のスカイライン数がほぼ最大値となっており，包含するサブシーケンスが現れることが少ないためである．OSR-RNT 法は，より PoI のカテゴリに偏りがあり，スカイライン数と最大数の差が大きい環境で，有効と考えられるが，本環境では，それほど有効ではない．

5.3 実 行 時 間

SSR 検索の実行時間を図 5 に示す．まず，図 5a より，h が大きくなると，スカイライン数の増加に加え，PoI の種類数も

増加するため，適合する PoI を探索するのに大きな時間がかかる．BSSR 法の実行時間は，h が 1 の場合を除いて，最も小さい．OSR 検索を複数回実行する場合では，同じ経路を何度も探索するが，BSSR 法では重複した探索を避けることができ，計算量が削減するからである．h が 1 のときは，SSR 検索は，OSR 検索と同様になるため，OSR 検索を 1 度実行するほうが早い．OSR-RCC 法の実行時間は，OSR-RNT 法にに比べ，半分程度の時間となっている．これは，上限値を更新していくことにより，計算量を大きく削減できていることを示す．一方で，OSR-RNT 法では，計算回数をそれほど削減できていないため，実行時間が大きくなっている．ダイクストラベース法と PNE 法を比較すると，h が 5 より小さい場合では，PNE 法の実行時間が小さいが，h が 5 の場合では，ダイクストラベース法の実行時間の方が小さい．これは，上限値が大きい場合，ダイクストラベース法の計算量の方が PNE 法より大きくなるが，上限値が小さくなってくるとダイクストラベース法の方が小さくなるためである．そのため，h が増加し，OSR 検索の探索範囲が広くなると，より上限値を有効活用するダイクストラベース法の方が，実行時間が短くなる．

次に，図 5b でも，同様に m が大きくなるにつれて，実行時間が大きくなるが，h の増加に比べると緩やかである．h が増加するとカテゴリ数が多くなるため，一度の OSR 検索の時間が大きく増加するが，m が増加してもカテゴリ数は多くならないため，計算時間もそこまで大きくならない．この結果においても，BSSR 法の実行時間は最も小さい．ダイクストラベース法を用いた OSR-RCC 法と OSR-RNT 法の実行時間は，h を変化させた結果と同様に倍程度の差がある．一方，PNE 法を用いた OSR-RCC 法と OSR-RNT 法の実行時間が，それほど大きな差がない．これは，PNE 法がそれほど上限値に大きく依存しないためである．

これらの結果より，一度に SSR 検索を実行する BSSR 法の性能が最もよいことがわかる．

6. おわりに

本稿では，新たな経路検索問合せとして，Skyline Sequenced Route (SSR) 検索を提案した．SSR 検索は，OSR 検索を拡張し，PoI と指定したカテゴリ集合のカテゴリ適合度という概念を用いて，経路に新たな評価指標を加えた．SSR 検索では，始点と順序つきのカテゴリ集合を指定すると，カテゴリを満たす PoI を順に満たす経路のうち，経路長とカテゴリ適合度の観点から，他の経路に支配されていない経路集合を求める．SSR 検索を効率的に計算する方法として，OSR-RCC 法，OSR-RNT 法，および Bulk SSR 法を提案した．OSR-RCC 法では，OSR 検索を繰り返しながら，経路長の上限値を更新していくことで，一度の OSR 検索の計算量を削減する．OSR-RNT 法では，求めた経路に包含されるカテゴリを含む経路を探索しないことで，OSR 検索の回数を削減する．Bulk SSR 法では，複数回の OSR 検索をせずに，一度の探索で SSR 検索を実現することにより，計算量を大きく削減する．実データを用いたシミュレーション実験により，Bulk SSR 法が最も実行時間時間が短いことを確認した．

現在オンライン検索のみで SSR 検索を実現しているが，今後は，インデックス技術や計算結果のキャッシュなどを用いることにより，さらなる計算量の削減を目指す．

謝　辞

本研究は独立行政法人科学技術振興機構（JST）の研究成果展開事業「センター・オブ・イノベーション（ＣＯＩ）プログラム」，および科学研究費（26540043, 25280039）の支援によって行われた．ここに記して謝意を表す．

文　献

[1] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[2] X. Cao, L. Chen, G. Cong, and X. Xiao. Keyword-aware optimal route search. *PVLDB*, 5(11):1136–1147, 2012.

[3] H. Chen, W.-S. Ku, M.-T. Sun, and R. Zimmermann. The multi-rule partial sequenced route query. In *GIS*, pages 1–10, 2008.

[4] F. Korn, N. Sidiropoulos, and C. Faloutsos. Fast nearest neighbor search in medical image databases. In *VLDB*, pages 215–226, 1996.

[5] H.-P. Kriegel, M. Renz, and M. Schubert. Route skyline queries: A multi-preference path planning approach. In *ICDE*, pages 261–272, 2010.

[6] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng. On trip planning queries in spatial databases. In *Advances in Spatial and Temporal Databases*, pages 273–290. Springer, 2005.

[7] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.

[8] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD*, pages 154–165, 1998.

[9] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. *The VLDB journal*, 17(4):765–787, 2008.

[10] M. Sharifzadeh and C. Shahabi. Processing optimal sequenced route queries using voronoi diagrams. *GeoInformatica*, 12(4):411–433, 2008.

[11] Y. Tian, K. C. Lee, and W.-C. Lee. Finding skyline paths in road networks. In *GIS*, pages 444–447, 2009.

[12] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang. Stochastic skyline route planning under time-varying uncertainty. In *ICDE*, pages 136–147, 2014.