時間依存道路ネットワークにおける各道路の最短移動時間に関するインデックスを用いたk最近傍検索手法

駒井 友香[1]　Nguyen Duong Hong[2]　原 隆浩[1]　西尾 章治郎[1]

概要：近年，各道路の移動時間が時刻により変化する時間依存道路ネットワークへの関心が高まっている．時間依存道路ネットワークにおける k 最近傍検索では，検索を行う時刻により，k 最近傍検索の検索結果が変化するため，従来の固定道路ネットワークにおける k 最近傍検索手法を直接適用することはできない．そこで本稿では，時間依存道路ネットワークにおいて，参照節点数および検索時間の削減を目的とする，各道路の最短移動時間に関するインデックスを用いた k 最近傍検索手法を提案する．提案手法では，前処理として，各節点から各データポイントまでの最短移動時間を表す，ヒューリスティック関数に基づくインデックスを作成する．検索時には，参照する節点におけるインデックスを参照し，これまでに検索されていないデータポイントのうち最も近いデータポイントまでの値を予想移動時間として用いることで，A* アルゴリズムを実行し，k 個のデータポイントを発見した時点で検索を終了する．シミュレーション実験より，提案手法が既存手法と比べ，参照する節点を削減し，検索時間を短縮できることを確認した．

1. 序論

近年，カーナビゲーションシステムやオンラインマップサービスが利用可能なスマートフォンやタブレットなどの携帯端末の普及に伴い，道路上のユーザの位置に応じた情報を提供する位置依存サービスが様々な場面で利用されている．このような位置依存サービスのうち，ユーザの周辺の飲食店や商業施設など (データポイント) の情報を取得するための有効な方法として，k 最近傍 (k-Nearest Neighbor，kNN) 検索がある．k 最近傍検索では，検索を行うユーザの位置（クエリポイント）から道路ネットワーク上の移動時間 (道路距離) が近い k 個のデータポイントの情報を取得する．k 最近傍検索を行うことにより，例えば，車内のユーザが道路距離の近い 10 個の店舗の情報やクーポンを効率的に取得することができる．

ここで近年，実際の交通状況を考慮した道路ネットワークとして注目されている，時間依存道路ネットワークに関する研究が盛んに行われている．この道路ネットワークは，各道路 (交差点間) の移動時間が，時間に依存する関数を用いて表される．図 1 はその一例であり，大阪市のある道路における車の平均移動時間の時間変化を表している．

このような時間依存道路ネットワークにおいて，k 最近傍検索を行う場合，固定道路ネットワークにおける k 最近傍検索とは異なる特徴がある．図 2 では，8 時において，ある車から最も近い店舗 (最近傍店舗) を検索する例を示し，図 3 では，同様の検索を 14 時に行う例を示している．それぞれの図において，各道路の傍に記されている時間は，それぞれの時刻における各道路の移動時間を示している．図 2 において，8 時には，店舗 1 への移動時間が 3 分と最も短く，店舗 1 が最近傍店舗となる．一方，図 3 では，店舗 2 への移動時間が 6 分と最も短く，14 時においては店舗 2 が最近傍店舗となる．このように，各道路の移動時間が時刻により変化するため，検索を行う時刻により k 最近傍検索の検索結果が変化する．そのため，時間依存道路ネットワークにおける k 最近傍検索に，従来の固定道路ネットワークにおける k 最近傍手法 [8] を直接適用することはできない．

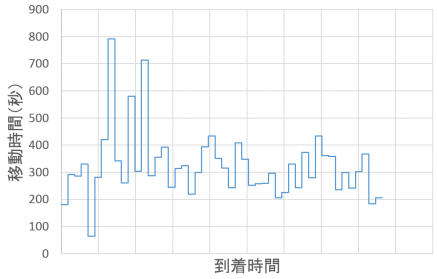


図 1　大阪市のある道路における移動時間の変化

これまでに，時間依存道路ネットワークにおける k 最近傍検索のための手法として，TD-INE 法 [2] が提案されて

[1] 大阪大学大学院情報科学研究科マルチメディア工学専攻
Yamadaoka 1-5, Suita, Osaka 565–0871, Japan
[2] 大阪大学工学部電子情報工学科

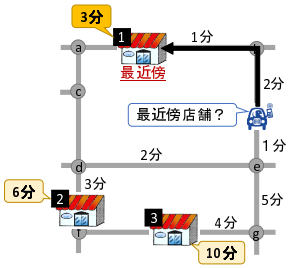


図 2 8 時の最近傍となる店舗

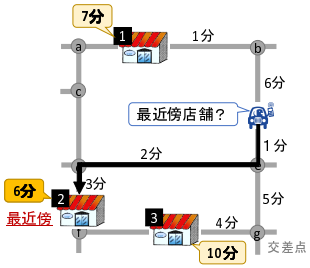


図 3 14 時の最近傍となる店舗

いる．TD-INE 法は，最短路検索手法であるダイクストラ法 [4] を基にしており，クエリポイントからの移動時間が小さい交差点 (節点) を順に探索し，k 個のデータポイントを発見した時点で検索を終える手法である．この方法では，検索を行う時刻の移動時間を参照しながら探索するため，各道路の移動時間の時間変化と関係なく，検索することができる．しかし，k 個のデータポイントよりも近いすべての節点を参照しなければならず，多くの無駄な節点を参照し，検索時間が大きくなってしまう．

ここで，道路ネットワーク上の最短路検索には，ダイクストラ法よりも効率のよい A*アルゴリズム [7] がよく用いられる．ある節点 (始点) からある節点 (終点) への最短路を検索する場合，A*アルゴリズムでは，探索する各節点において，その節点までの移動時間に，終点 (目的地) までの予想移動時間 (ヒューリスティック関数) を加えた値を算出し，この値が最小となる隣接節点を次の参照節点とする．これにより，全方向に参照節点を拡張するダイクストラ法と比べ，終点に向けて探索を行うことができ，参照する節点を少なくすることができる．

この A*アルゴリズムを，時間依存道路ネットワークにおける k 最近傍検索に適用すると，ダイクストラ法を利用した従来手法である TD-INE 法より，少ない参照節点および短い検索時間を達成できるものと考えられる．ただし，その効果の度合は A*アルゴリズムで用いられるヒューリスティック関数の定義に大きく依存するものと予測される．時間依存道路ネットワークでは，時刻により道路の移動時間が変化するため，移動時間の予測が，固定道路ネットワークに比べ困難である．また，A*アルゴリズムは最短路検索手法であるため，終点が既知である必要がある．しかし，k 最近傍検索では，終点となるデータポイント自体が検索の対象 (どのデータポイントが k 最近傍であるかが未知) であるため，全てのデータポイントへの最短路を検索しなければ，結果を得ることができない．そのため，多くの節点を参照することとなり，検索に時間がかかってしまう．

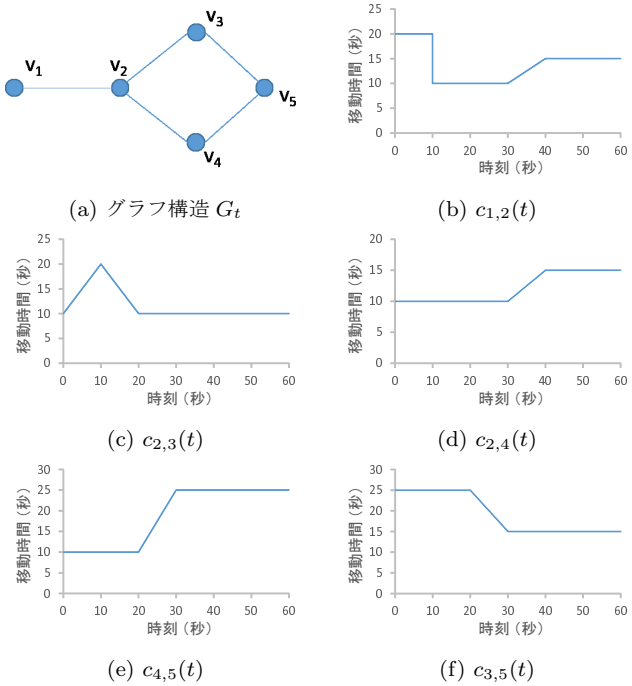
そこで本稿では，時間依存道路ネットワークにおける参照節点数および検索時間の削減を目的として，各道路の最短移動時間に関するインデックスと A*アルゴリズムを用いた k 最近傍検索手法を提案する．提案手法では，前処理として，ヒューリスティック関数に基づくインデックスを作成する．この際，2 種類の異なるヒューリスティック関数を用いる．一方は，全時間帯における移動時間の最短値を，各道路の移動時間としたものであり，もう一方は，全時間をある時間幅で区切り，その時間帯における移動時間の最短値を，各道路の移動時間としたものである．インデックスの項目としては，各節点から近い C 個のデータポイントおよびそのデータポイントまでの最短移動時間を保持する．検索時には，参照する節点におけるインデックスのうち，検索時刻が該当する時間帯のインデックスを参照し，これまでに検索されていないデータポイントのうち最も近いデータポイントまでの値を予想移動時間として用いることで，A*アルゴリズムを実行する．これをネットワーク内の探索範囲を拡張しながら繰り返し，k 個のデータポイントを発見した時点で検索を終了する．提案手法では，k 最近傍となる可能性の高いデータポイントへ向けて，検索を行うことができるため，参照節点数を削減することができる．また，ヒューリスティック関数に基づくインデックスを参照することで，検索時間を削減することができる．

以下では，2. で問題定義を行い，3. で関連研究について述べる．その後，4. で提案手法について説明し，5. でシミュレーション実験の結果を示す．最後に，6. で本稿のまとめと今後の課題について述べる．

2. 問題定義

本章では，時間依存道路ネットワークにおける k 最近傍検索問題を定義する．本稿では，道路ネットワークに固定のデータポイントが多数存在する環境を想定し，ユーザは，これらのデータポイントを検索するクエリを発行するものと想定する．また，簡単化のため，クエリを発行する位置 (クエリポイント) に対する k 最近傍検索を行う．本稿で想定する時間依存道路ネットワークは，各道路を辺，各交差点を節点とする重み付きグラフモデルと見なせる．各辺の重みは時刻で変化し，ある時刻の重み (移動時間) は，統計値などにより既知であるとする．また，データポイントはグラフ上の節点として扱う．以下に，時間依存道路グラフ $G_t(V, E)$ を定義する．

定義 2.1: 時間依存道路グラフ $G_t(V, E)$. 時間依存道路グラフ G_t は $G_t(V, E)$ として定義される．ここで，$V = \{v_1, v_2, ..., v_m\}$ は節点の集合であり，$E (E \subseteq V \times V)$ は二つの節点を接続する辺の集合である．各辺 e は $e(v_i, v_j)$ で示される．G_t は無向グラフであり，$e(v_i, v_j) = e(v_j, v_i)$ である．各辺 $e(v_i, v_j)$ は時刻 t に v_i から v_j まで移動する時間 $c_{v_i,v_j}(t)$ を保持している．図 4(a) は 5 つの節点 v_1, v_2, v_3, v_4, および v_5 からなる時間依存道路グラフを表しており，図 4(b) から (f) は，各辺における移動時間の時刻による変化を示している．ここで，各辺 $e(v_i, v_j)$ に対して，v_i から v_j までの移動時間が全時刻で最も小さい値を



(a) グラフ構造 G_t　(b) $c_{1,2}(t)$

(c) $c_{2,3}(t)$　(d) $c_{2,4}(t)$

(e) $c_{4,5}(t)$　(f) $c_{3,5}(t)$

図 4　時間依存グラフ

$min(c_{v_i,v_j})$ と定義する．また，各辺 $e(v_i, v_j)$ に対して，v_i から v_j までの移動時間が時刻 t から t' の間で最も小さい値を $min(c_{v_i,v_j}[t, t'])$ として定義する．

定義 2.2: 経路の移動時間．経路 $\{s = v_1, v_2, ..., v_j = d\}$ を節点を通過する順序 (経路) として表す．ここで，$e(v_i, v_{i+1}) \in E$ および $i = 1, ..., j-1$ である．また，s は経路の始点となる節点であり，d は経路の終点となる節点である．G_t 上の経路では，各辺の移動時間は，その辺に到着する時刻に応じて変化するため，時刻 t_s における s から d までの経路の移動時間は以下のように計算される．

$$TT(s \longrightarrow d, t_s) = \sum_{i=1}^{j-1} c_{v_i,v_{i+1}}(t_i) \qquad (1)$$

ここで，$t_1 = t_s, t_{i+1} = t_i + c_{v_i,v_{i+1}}(t_i), i = 1, ..., j-1$ である．

定義 2.3: 時間依存最短経路．G_t において，時刻 t_s に s から d までの移動時間が最も小さくなる経路を $TDFP(s, d, t_s)$ と定義する．例えば，$t_s = 5$ のとき，図 4(a) において，v_1 から v_5 までの $TDFP$ は，$TDFP(v_1, v_5, 5) = \{v_1, v_2, v_3, v_5\}$ である．

時間依存最短経路 $TDFP(s, d, t_s)$ に対して，時間依存最短移動時間 $TDFT(s \longrightarrow d, t_s)$ は以下のように計算される．

$$TDFT(s \longrightarrow d, t_s) = TT(s \longrightarrow d, t_s) \qquad (2)$$

定義 2.4: 時間依存 k 最近傍検索．時間依存 k 最近傍検索は，時間依存ネットワークにおいて，クエリポイントから移動時間の最も短い k 個のデータポイントを検索するクエリとして定義される．

3. 関連研究

3.1 時間依存グラフにおける最短路検索

k 最近傍検索の結果は，クエリポイントから各データポイントまでに移動する最短経路 (または最速経路) のコスト (移動時間) に依存するため，最短路検索は，k 最近傍検索に関連した問題である．文献 [5] において，時間依存最短路問題は，ダイクストラ法の一般化によって解決できることを示している．ただし，文献 [6] において，ダイクストラ法の一般化が，対応できるのは各移動オブジェクトの道路に出る順序とその道路に入る順序が変わらない First-in-first-out(FIFO) 時間依存道路ネットワークのみであることが証明されている．また，文献 [3] では，道路ネットワーク分割による階層化を用いて，ヒューリスティック関数を計算し，双方向 A*アルゴリズムを用いて，最短路検索を行う手法を提案している．これら最短路検索は，終点となるデータポイントへの最短路を取得するものであり，本研究で想定する k 最近傍検索とは取り扱う問題が異なるため，そのまま適用することはできない．

3.2 時間依存道路ネットワークにおける k 最近傍検索

文献 [2] において，時間依存道路ネットワークにおける k 最近傍検索が初めて提唱されており，二つの手法が提案されている．一つ目の手法では，ネットワークを時間拡張グラフにモデル化し，固定道路ネットワークの手法を利用する．しかし，この手法は検索時間が増加する問題，および正確な結果を保証しないという問題がある．二つ目の手法は，文献 [8] で提案された INE 法を時間依存道路ネットワークに適用している手法であり，k 個のデータポイントを発見するまでダイクストラ法により順に節点を参照する．

文献 [1] において，二つの異なるインデックス構造である，Tight Network Index(TNI) および Loose Network Index(LNI) を用いる k 最近傍検索手法が提案されている．TNI は，各道路の最短移動時間を用い，データポイントごとに，時刻に依存することなく必ずそのデータポイントが最近傍となる節点の最短路木から構築されている．また，LNI は，各道路の最長移動時間を用い，データポイントごとに，ある時刻でそのデータポイントが最近傍となる節点の最短路木から構築されている．これらのインデックスを用い，k 最近傍となるデータポイントの候補を絞り，候補となったデータポイントまでの移動時間を計算することにより，最終的な結果を取得する．しかし，検索時にクエリポイントから k 最近傍データポイントの候補となったデータポイントまでの最短経路を計算しなければならず，必ずしも検索時間が短くなるとは限らない．また，データポイントの密度が大きい場合，k 最近傍の候補となるデータポイントの数が多くなり，検索時間が増加する．

4. 提案手法

本章では，まず手法の設計方針について述べ，その後，提案するヒューリスティック関数について説明する．最後に，提案手法である FTT インデックスによる k 最近傍検索および TD-FTT インデックスによる k 最近傍検索について, インデックス作成方法および検索手法をそれぞれ説明する．

4.1 設計方針

時間依存道路ネットワークでは，各データポイントまで移動する時間が時刻によって変わるため，k 最近傍検索の検索結果も時刻に応じて変化する．そのため，固定道路ネットワークにおける手法でよく用いられるような，最短路を事前に計算しインデックスとして保存するアプローチでは，時刻ごとにその情報を保存するための膨大な記憶領域が必要であり，非効率である．そのため，提案手法では，節点から節点へのネットワーク拡張により k 最近傍検索を行う．その際，無駄な節点の参照を抑制するため，ネットワーク拡張の領域を絞り込む必要がある．提案手法では，A*アルゴリズムを用いることにより，これを実現する．

A*アルゴリズムでは，始点となる節点から，ある節点 n を経由し，終点となる節点までの最短経路を探索する．このとき，最短経路のコスト移動時間を $f(n)$ とすると，$f(n) = g(n) + h(n)$ とできる．ここで, $g(n)$ は始点から n までの最小コスト, $h(n)$ は n から終点までの最小コストである．もし $g(n)$ の値と $h(n)$ の値が既知であれば, 全体の最短経路 $f(n)$ は容易に求まる．

時間依存道路ネットワークでは，$g(n)$ は探索の過程で求めることができるが，$h(n)$ は事前に正確な値を把握することは困難である．そこで，$f(n)$ を $f^*(n) = g(n) + h^*(n)$ とする．$h^*(n)$ は n から終点までの最小コストの推定値である．$h^*(n)$ をヒューリスティック関数とよぶ．A*アルゴリズムでは，$h^*(n)$ が，$\forall n, 0 \leq h^*(n) \leq h(n)$ を満たすとき、求まる経路が始点から終点までの最短経路であることが保証されている．また, $\forall n, h_1^*(n) < h_2^*(n) \leq h(n)$ となるヒューリスティック関数が存在する場合は h_1^* を用いるよりも h_2^* を用いた方が，検索範囲を抑制できる．提案手法は，この A*アルゴリズムを用いることにより，ネットワークの全方向にグラフを拡張するダイクストラ法と異なり，目的のデータポイントに向かう経路のみを効率的に探索する．この際，上述のように，最短路検索を保証するためのヒューリスティック関数を採用する必要がある．さらに, 上述のように, ヒューリスティック関数の返す値が，検索を行う時刻における移動時間 (コスト) に近いほど，参照する節点の数を削減できる．そのため, 実際の移動時間に近い値を返すヒューリスティック関数を検討する．

4.2 提案ヒューリスティック関数

4.2.1 最短移動時間ヒューリスティック関数

定義 4.1: 最短移動時間道路ネットワーク.

時間依存グラフ $G_t(V, E)$ の最短移動時間道路ネットワークを $G_{FTT}(V, E)$ と定義する．ここで, V および E の集合は $G_t(V, E)$ と同じであるが, 各辺 $e(v_i, v_j)$ の重みは $min(c_{v_i,v_j})$ である．$FP(s, d)$ は $G_{FTT}(V, E)$ における s から d までの最短路と定義する．経路 $FP(s, d) = \{s = v_1, v_2, ..., v_j = d\}$ であり，節点は通過した順に示す．また，全時間での s から d までの最短移動時間は，以下のように計算できる．

$$FTT(s \longrightarrow d) = \sum_{i=1}^{j-1} min(c_{v_i,v_{i+1}}) \tag{3}$$

G_t 内の任意の経路に対して，$FTT(s \longrightarrow d) \leq TDFT(s \longrightarrow d, t_s)$ である．

定義 4.2: 最短移動時間ヒューリスティック関数. 道路ネットワーク上のデータポイントの集合 D を $\{d_1, d_2, ..., d_x\}$ と定義する．クエリポイント q からネットワーク拡張するとき, ある節点 n を考える． このとき，既に発見された検索結果に入るデータポイントの集合 NN を $\{NN_1, NN_2, ..., NN_l\}$ $(l < k)$ と定義する．節点 n において，終点 $d(n)$ はデータポイント集合 $D \backslash NN$ のうち, $G_{FTT}(V, E)$ において, n から最近傍となるデータポイントと定義する．ここで, $(d(n) \in D \backslash NN)$ である．つまり，$h^*(n) = FTT(n \longrightarrow d(n))$ である．このヒューリスティック関数が，正確に k 最近傍となるデータポイントを探索できることを証明する．

証明: このヒューリスティック関数が, 正確に, k 最近傍となるデータポイントを取得できることを証明するため, クエリポイント q からネットワーク拡張中に発見されたデータポイントまでの経路は最短路であること, および, 移動時間の小さい順にデータポイントを取得することを証明する. 時間依存道路ネットワークにおける k 最近傍となるデータポイントの集合を $\{NN_1, NN_2, ..., NN_k\}$ とする. クエリポイント q から i 番目に近いデータポイント NN_i までの経路上の節点 n' を考える．このとき，$\{h^*(n') = FTT(n' \longrightarrow d(n')) \leq FTT(n' \longrightarrow NN_i) \leq TDFT(n' \longrightarrow NN_i, t_{n'}) =$ (n'から NN_iまでの実際の移動時間)$\}$ となる．ここで, $t_{n'}$ は n' に到着する時刻である．つまり，$h^*(n')$ は，A*アルゴリズムのヒューリスティック関数の条件を満たすため, クエリポイントから NN_i までの経路は最短路であることを保証できる．また, データポイントは, $h^*(n') + TDFT(q, n')$ が小さい順に選ばれるため, 先に発見されたデータポイントは，まだ参照されていないデータポイントよりも，クエリポイントから近いことを保証できる． □

ここで，全時間帯の最小移動時間を用いると，A*アルゴリズムにおける最短路の探索は保証できるが，予想移動時

間を実際の移動時間より大幅に小さく見積もってしまうことが多く，無駄に節点を参照してしまう可能性が高い．そこで，時間帯ごとの最短移動時間を用いることにより，この問題を解決する．

4.2.2 時間帯ごとの最短移動時間ヒューリスティック関数

定義 4.3: 時間帯 $[t,t']$ の最短移動時間道路ネットワーク．

ある時間帯 $[t,t']$ における，時間依存グラフ $G_t(V,E)$ の最短移動時間道路ネットワークを $G_{FTT}[t,t'](V,E)$ と定義する．ここで，V および E の集合は $G_t(V,E)$ と同様であるが，各辺 $e(v_i,v_j)$ の重みは $min(c_{v_i,v_j}[t,t'])$ である．$FP[t,t'](s,d)$ を $G_{FTT}[t,t'](V,E)$ における s から d までの最短路と定義する．経路 $FP[t,t'](s,d) = \{s = v_1, v_2, ..., v_j = d\}$ であり，節点は通過した順に示す．また，時間帯 $[t,t']$ における，s から d までの最短移動時間は以下のように計算される．

$$FTT[t,t'](s \longrightarrow d) = \sum_{i=1}^{j-1} min(c_{v_i,v_{i+1}}[t,t']) \qquad (4)$$

また，$t_s \in \{t,t'\}$ のとき，$FTT[t,t'](s \longrightarrow d) \leq TDFT(s \longrightarrow d, t_s)$ である．

定義 4.4: $[t,t']$ における最短移動時時間ヒューリスティック関数．クエリ発行時刻は $t_s(t_s \in [t,t'])$ とする．t_s に発行されたクエリポイントから k 最近傍となるデータポイントまでの移動時間は時間帯 $[t,t']$ を超えないと想定する．道路ネットワーク上のデータポイントの集合 D は $\{d_1, d_2, ..., d_x\}$ と定義する．t_s に発行されたクエリポイント q からネットワーク拡張するとき，ある節点 n を考える．このとき，既に発見された検索結果に入るデータポイントの集合 NN は $\{NN_1, NN_2, ..., NN_l\}$ $(l < k)$ として定義する．また，節点 n において，終点 $d(n)$ はデータポイント集合 $D \setminus NN$ のうち，$G_{FTT}[t,t'](V,E)$ において，n から最近傍となるとなるデータポイントと定義する．ここで，$d(n) \in D \setminus NN$ である．つまり，$h^*(n) = FTT[t,t'](n \longrightarrow d(n))$ である．このヒューリスティック関数は，正確に k 最近傍となるデータポイントを探索できることを証明する．

証明: t_s に発行されたクエリポイントから k 最近傍となるデータポイントまでの移動時間は時間帯 $[t,t']$ を超えない想定の下で，定義 4.2 における証明と同様である．□

4.3 インデックスを用いた k 最近傍検索手法

提案手法は，前節で説明したヒューリスティック関数を用いた A*アルゴリズムにより，時間依存道路ネットワークにおいて，無駄な節点を参照することなく k 最近傍検索を行うことができる．ここで，k 最近傍検索中にヒューリスティック関数を計算すると計算コストが増大し，検索時間が大きくなってしまう．そのため，ヒューリスティック関数が返す値を事前に計算し，これを格納するインデックス構造を提案する．これにより，高速に A*アルゴリズムを実行でき，検索結果を速く取得できる．

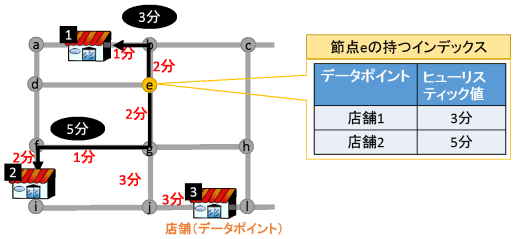


図 5 FTT インデックス作成の例

4.3.1 Fast Travel Time インデックス

本項では，最短移動時間ヒューリスティック関数に関する Fast Travel Time(FTT) インデックス構造について説明する．このインデックスは，節点ごとに保持する．

前処理: まず，$G_{FTT}(V,E)$ において，各節点は，自身の位置をクエリポイントとした場合の C 最近傍となるデータポイントをダイクストラ法を用いて計算する．このとき，そのデータポイントまでの移動時間も計算する．これらの値を移動時間が小さい順にインデックスに保持する．

図 5 を用いて，節点 e のインデックス作成の例を示す．図は $G_{FTT}(V,E)$ の例であり，ここでは，データポイントは店舗で表される．道路 (b,e), $(b,$店舗 1$)$, (e,g), (g,f), $(f,$店舗 2$)$ の最短移動時間はそれぞれ 2 分, 1 分, 2 分, 1 分, 2 分である．$G_{FTT}(V,E)$ における節点 e の最近傍となる店舗は店舗 1 であり，2 最近傍となる店舗は店舗 2 である．この例において，$C = 2$ とする．この場合，節点 e のインデックスには，店舗 1 と店舗 2 および e からそれらの店舗までの移動時間 (ヒューリスティック値) が保持される．

k 最近傍検索: 提案手法における k 最近傍検索は，インデックスに格納したヒューリスティック値 (予想移動時間) を参照し，A*アルゴリズムを用いることにより実行する．検索中に参照された節点は，インデックス中に格納されている (その節点からの) 最近傍から C 最近傍となるデータポイントのうち，検索中に参照されていないデータポイントを調べる．参照されていないデータポイントにおいて，最も近いデータポイントに対応するヒューリスティック値を予想移動時間 $(h^*(n))$ として利用する．この操作を k 個のデータポイントを発見するまで繰り返す．提案手法の k 最近傍検索アルゴリズムをアルゴリズム 1 に示す．

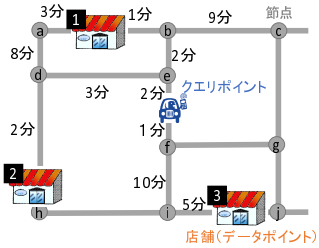
図 6 を用いて，2 最近傍検索の例を説明する．まず，クエリポイント q からクエリポイントに隣接する節点 e と節点 g まで拡張し，スタックに節点 e と節点 g およびクエリポイントから節点 e と節点 g までの移動時間を格納する．節点 e において，節点 e に対するインデックスを参照し，店舗 1 が最短のヒューリスティック値をもつため，この値を用いて A*アルゴリズムのコスト値を計算する．節点 g では，店舗 3 が最短となるため，この移動時間を用いて，節点 e と同様の操作を行なう．節点 e のコストは 2 分+3 分 =5 分，節点 g のコストは 1 分+9 分=10 分である．スタッ

Algorithm 1 kNN(q, $start\_time$, $index$,k)

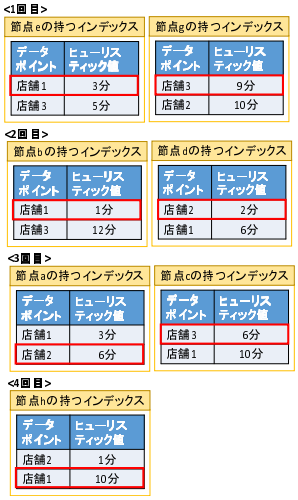
// q:クエリポイント
// $start\_time$: 検索の開始時刻
// $come\_time$: 道路に到着する時刻
// $index$:FTT インデックス
// k: 検索するデータポイントの数
// u: 参照している節点
// $vertex$: 節点の集合
// $vertex[i]$: 節点 i
// $vertex[i].travel\_time$: クエリポイントから節点 i までの移動時間
// $vertex[i].label = 0$: 節点 i をスタックに入れない
// $vertex[i].label = 1$: 節点 i をスタッに入れた
// $vertex[i].cost$: $vertex[i].travel\_time + FTT(n \longrightarrow d(n))$
// $vertex[i].pointer$: 節点 i のインデックスにおけるヒューリスティック値に指定しているポインタ
// $vertex[i].adjacent$: 節点 i の隣接する節点の集合
// $stack$: 参照している節点のスタック
// $sort(stack)$: スタックにある節点を $cost$ が大きい順に並ぶ
// $stack.insert(vertex[i])$: 節点 i をスタックに入れる
// $TD(road\_id, come\_time)$: 道路識別 $road\_id$ およびその道路の到着時間 $come\_time$ から道路の移動時間を返す関数
// $Datapoint$: データポイントの集合
// $Result$: k 最近傍となるデータポイントの集合
// $Result.insert(u)$: データポイント u を集合 $Result$ に入れる

1: **for** $i = 1$ to n **do**
2: $vertex[i].travel\_time = \infty$;
3: $vertex[i].label = 0$;
4: **end for**
5: $q.label = 1$;
6: $q.travel\_time = 0$;
7: $q.cost = index[q].travel\_time[0]$;
8: $stack.insert(q)$;
9: **while** $stack$ is not empty and $Result.size() < k$ **do**
10: $sort(stack)$;
11: $u \leftarrow stack.pop$;
12: $come\_time = u.travel\_time + start\_time$;
13: **for** $j = 1$ to $u.adjacent.size()$ **do**
14: $travel\_time = TD(e(u, u.adjacent[j]), come\_time) + u.travel\_time$;
15: **if** $travel\_time < u.adjacent[j].travel\_time$ **then**
16: $u.adjacent[j].travel\_time = travel\_time$;
17: **if** $u.adjacent[j].pointer.rank < c$ **then**
18: **while** $u.adjacent[j].pointer.data \in Result$ **do**
19: $u.adjacent[j].pointer = u.adjacent[j].pointer + 1$;
20: **end while**
21: **end if**
22: $u.adjacent[j].cost = travel\_time + u.adjacent[j].pointer.heuristic\_value$;
23: **if** $u.adjacent[j].label \neq 1$ **then**
24: $u.adjacent[j].label = 1$;
25: $stack.insert(u.adjacent[j])$;
26: **end if**
27: **end if**
28: **end for**
29: **if** $u \in Datapoint$ **then**
30: $Result.insert(u)$;
31: **end if**
32: **end while**
33: **return** $Result$

ク中の，最小コストを保持する節点は e であるため，e をスタックからポップし，節点 e に隣接する節点 b と節点 d をスタックに加える．次に，(e, d) の移動時間は 3 分であるため，(q, d) の移動時間は 5 分である．節点 d における最近傍の店舗は店舗 2 であるため，ヒューリスティック値として 2 分を用いる．そのため，節点 d のコストは 5 分+2 分=7 分である．同様にして，b のコストは 5 分となる．このとき，スタック中の最小コストをもつ節点は b となり，b をポップして，節点 b に隣接する節点 a と節点 c をスタックに加える．その結果，道路 (b, a) 上に最近傍となる店舗 1 を発見できる．2 最近傍となるデータポイントを検索する際も，上記の操作を繰り返す．このとき，すでに検索済みの店舗 1 に対応するインデックスは参照しない．これにより，2 最近傍となるデータポイントである店舗 2 を検索でき，検索を終了する．



(a) 道路ネットワーク



(b) インデックス参照課程

(c) スタック

図 6 2 最近傍検索例

4.3.2 Time Dependent Fast Travel Time インデックス

本節では，最短移動時間ヒューリスティック関数に基づく Time Dependent Fast Travel Time(TD-FTT) インデックスについて説明する．TD-FTT は，時間帯ごとに FTT を構築するインデックスである．

前処理: 時間幅 $[0, t_b]$ を各時間帯 $[0, t_1]$, $[t_1, t_2]$,..., $[t_{b-1}, t_b]$ に区切る． ある時間帯 $[t_i, t_{i+1}]$ において，最短移動時間固定グラフ $G_{FTT}[t_i, t_{i+1}](V, E)$ を作成する． この $G_{FTT}[t_i, t_{i+1}](V, E)$ において，FTT を作成する．これを全ての時間帯で実行することにより，TD-FTT が作成される．

k 最近傍検索アルゴリズム: k 最近傍検索も，4.3.1 項と基本的に同様の方法で実行される．しかし，各節点は時間帯ごとのインデックスを保持しているため，クエリの発行時刻によって参照するインデックスが異なる．例えば，クエリが発行された時刻が t である場合，各節点が参照するインデックスは，$t \in [t_i, t_{i+1}]$ となる時間帯のものである．

図 7 に, TD-FTT インデックスを参照する例を表す．時

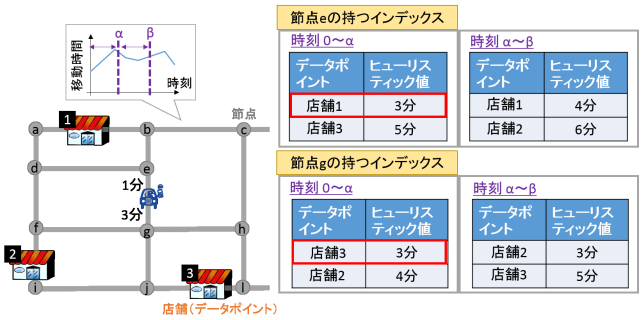


図 7　TD-FTT インデックスの参照

間帯 $[0, \alpha]$ 内に検索を行う場合，節点 e では，時間帯 $[0, \alpha]$ のインデックスを参照し，店舗 1 まで予想する移動時間である 3 分をヒューリスティック値として使う．節点 g では，時間帯 $[0, \alpha]$ のインデックスを参照し，店舗 3 まで予想する移動時間である 3 分をヒューリスティック値として使う．

5.　性能評価

5.1　評価環境

本章では，提案手法の有効性を検証するために行ったシミュレーション実験の結果を示す．本実験は Core i7 (3.4GHz) および 8GB RAM を装備したコンピュータ上で実行した．また，評価した全ての手法は C++により実装した．ネットワークグラフの管理には，Boost Graph Library 1.54.0 [*1]を利用した．また，実際の環境での性能を確認するため，全国デジタル道路地図データベースから大阪市の地図を用いて評価を行った．具体的には，第二次地域メッシュコード 523503 と 523504 に相当する範囲 (約 10km × 20km 四方) であり，交差点の数は 14,027，道路の数は 21,189 である．このグラフ上に，データポイントを一様に配置した．

本田技研工業株式会社[*2]より提供された同範囲内の車両トレースデータを用いて，9 時から 18 時までの時間依存道路ネットワークを作成した．具体的には，車両トレースデータを前述の地図上の道路とマッチングし，各車両の道路上の滞在時間を，その車両が道路に進入した時刻における移動時間とした．各道路において，各車両の移動時間を 1 時間ごとに平均したものを時間依存道路グラフの各辺の重みとした．この際，車両が 1 台も通過しない，もしくは 1 度だけ通過した道路は，車両が通過した数が最も多い道路における移動時間の時間変化をモデル化したものを適用した．

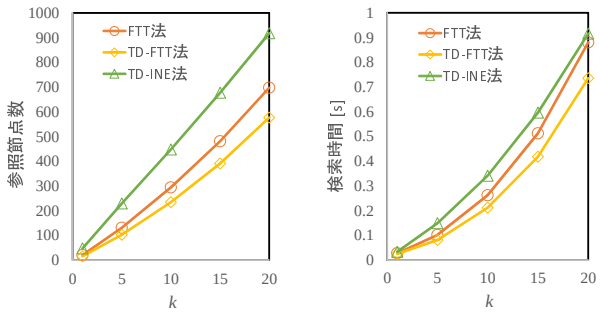
比較手法として TD-INE 法 [2] を用いた．提案手法は，FTT インデックスを用いるものを FTT 法，TD-FTT インデックスを用いるものは TD-FTT 法と記す．本実験において，FTT インデックスでは，各道路において，9 時から 18 時までの最短移動時間を用いた．TD-FTT インデックスでは，9 時から 18 時を 3 時間ごとの 3 つの時間帯に区切り，それぞれの時間帯において道路の最短移動時間を取り，3 つの時間帯のインデックスを作成した．

[*1] *http://www.boost.org/doc/libs/1_54_0/libs/graph/doc/index.html*

[*2] *http://www.honda.co.jp/*

表 1　パラメータ設定

パラメータ	基本値	範囲
データポイントの数	300	100, 200, 300, 400, 500
k	10	1, 5, 10, 15, 20



(a) 参照節点数　　(b) 検索時間

図 8　k の影響

表に本実験で用いたパラメータを示す．各パラメータは基本的には定数値をとるが，そのパラメータの影響を調べる際には表中の右側の範囲で変化させた．一つのインデックスに格納するデータポイント数 (C) は 20 とした．

以上のシミュレーション環境において，クエリポイントをランダムに選択し，100 回の検索を行った場合の，以下の評価値を調べた．

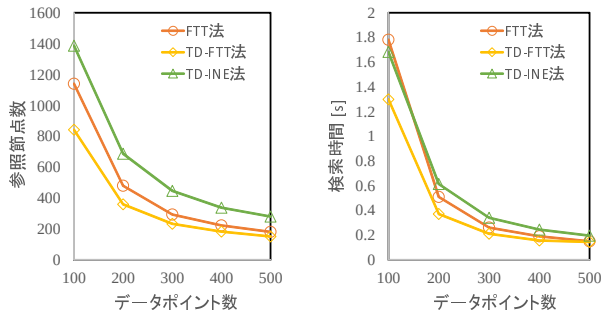
- 参照節点数: 検索結果を得るまでに参照した節点の数の平均
- 検索時間: 検索を開始してから検索結果を得るまでの平均時間 [s]

5.2　k の影響

検索するデータポイント数 k を変化させたときの結果を，図 8 に示す．

図 8(a) において，提案手法 (FTT 法および TD-FTT 法) が TD-INE 法より参照節点数が少ないことがわかる．k が大きくなると，参照した節点の数の差も増加する．これは，FTT 法および TD-FTT 法では，A*アルゴリズムを用いるため，基本的に，k 最近傍となるデータポイントに向かう経路中に存在する節点のみ参照しており，無駄な節点の参照が少ないためである．さらに，TD-FTT 法の参照節点数は，FTT 法よりも小さい．これは，TD-FTT 法で用いるヒューリスティック関数 (予想移動時間) は，FTT 法で用いる値よりも，実際の移動時間に近い値であるため，無駄な参照をさらに削減できるからである．

図 8(b) より k が 1 のとき，手法間の性能差は小さい．これは，最近傍となるデータポイントがクエリポイントから非常に近い範囲に存在するので，どの手法を用いても，最短路を速く計算できるためである．k が大きい場合は，提



(a) 参照節点数　　(b) 検索時間

図 9　データポイント数の影響

案手法は TD-INE 手法よりも検索時間が短い．ここで，参照節点数に対して検索時間の差が小さいのは，提案手法ではインデックスを参照するコストが発生するためである．また，TD-FTT 法の検索時間は，FTT 法に比べて小さい．これは，参照節点数が小さく，インデックスを参照するコストも小さくなるためである．

5.3　データポイント数の影響

道路ネットワーク内の全データポイント数を変化させたときの結果を, 図 9 に示す．

図 9(a) より，データポイントの数が小さい場合，提案手法は TD-INE 法に比べて参照節点数が小さいことがわかる．これは，TD-INE 法はダイクストラ法を用いているため，データポイントの数が小さい (データポイントの密度が小さい) 場合，k 最近傍となるデータポイントを発見するまでに大きな範囲を全方向へ検索するため，無駄な節点の参照が増加してしまうためである．一方，提案手法は A*アルゴリズムにより，この問題を抑制できている．

図 9(b) より，データポイントの数が多い場合，k 最近傍となるデータポイントは，クエリポイントの周辺に存在する場合が多くなり，検索時間が短くなる．ここで，k が 200 以上の場合，提案手法は TD-INE 手法よりも検索時間が短く，また，手法間の検索時間の差は，参照節点数の差に依存していない．これは 5.2 節で述べた理由と同様である．一方，k が 100 の場合，FTT 法は TD-INE 法よりも検索時間が大きくなっている．これは，多くの節点を参照する場合，提案手法ではインデックスの参照に大きな時間がかかってしまうためである．TD-FTT 法においては，参照節点数が小さく，検索時間を短くできていることから，時間帯で分割し，より実際の移動時間に近いヒューリスティック値をインデックスとして保持することが有効であることがわかる．

6.　結論

本稿では，時間依存道路ネットワークにおいて，k 最近傍検索の検索時間および参照節点数の削減を目的とする手法を提案した．提案手法では，FTT インデックスおよび TD-FTT インデックスと A*アルゴリズムを用いて，k 最近傍となるデータポイントを探索する．FTT インデックスは，各道路の全時間における最短移動時間を用いて，各節点において計算したヒューリスティック値を保持する．TD-FTT インデックスは，より効率よく A*アルゴリズムを実行するため，全時間をある時間幅で分割し, 各節点において，時間帯ごとに各道路の最短移動時間を用いたヒューリスティック値を保持する．道路ネットワークを拡張しながら k 最近傍検索時には，これらのインデックスを参照して，A*アルゴリズムにより, 道路ネットワークを拡張しながら k 最近傍となるデータポイントを発見する．シミュレーション実験の結果から，提案手法では，従来のダイクストラ法を用いた手法よりも参照する節点の数を抑制し，多くの場合で検索時間が短いことを確認した．また，TD-FTT インデックスを用いることで，FTT インデックスを用いた場合よりも，参照節点数および検索時間を削減できることを確認した．今後は時間依存道路ネットワークにおいて，連続的に k 最近傍検索を行う状況を想定し, 提案手法の拡張を行う予定である．

謝辞

本研究の一部は，文部科学省科学研究費補助金・基盤研究 B(24300037) および基盤研究 A(2620013) の研究助成によるものである．ここに記して謝意を表す．

参考文献

[1] U. Demiryurek, F. Banaei-Kashani, and C. Shahabi, “Efficient K-nearest neighbor search in time-dependent spatial networks,” *Proc. Int’l Conf. on Database and Expert Systems Applications*, vol.1, pp.432-449, 2010.

[2] U. Demiryurek, F. Banaei-Kashani, and C. Shahabi, “Towards K-nearest neighbor search in time-dependent spatial network databases,” *Databases in Networked Information Systems*, vol.5999, pp.296-310, 2010.

[3] U. Demiryurek, F. Banaei-Kashani, C. Shahabi, and A. Ranganathan, “Online computation of fastest path in time-dependent spatial networks,” *Proc. Int’l Conf. on Advances in Spatial and Temporal Databases*, pp.92-111, 2011.

[4] E. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol.1, no.1, pp.269-271, 1959.

[5] S.E. Dreyfus, “An appraisal of some shortest-path algorithms,” *Operations Research*, vol.17, no.3, pp.395-412, 1969.

[6] J. Halpern, “Shortest route with time dependent length of edges and limited delay possibilities in nodes,” *Zeitschrift fur Operations Research*, vol.21, no.3, pp.117-124, 1977.

[7] P. Hart, N. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Trans. on Systems Science and Cybernetics*, vol.4, no.2, pp.100-107, 1968.

[8] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, “Query processing in spatial network databases,” *Proc. Int’l Conf. on Very Large Data Bases*, vol.29, pp.802-813, 2003.